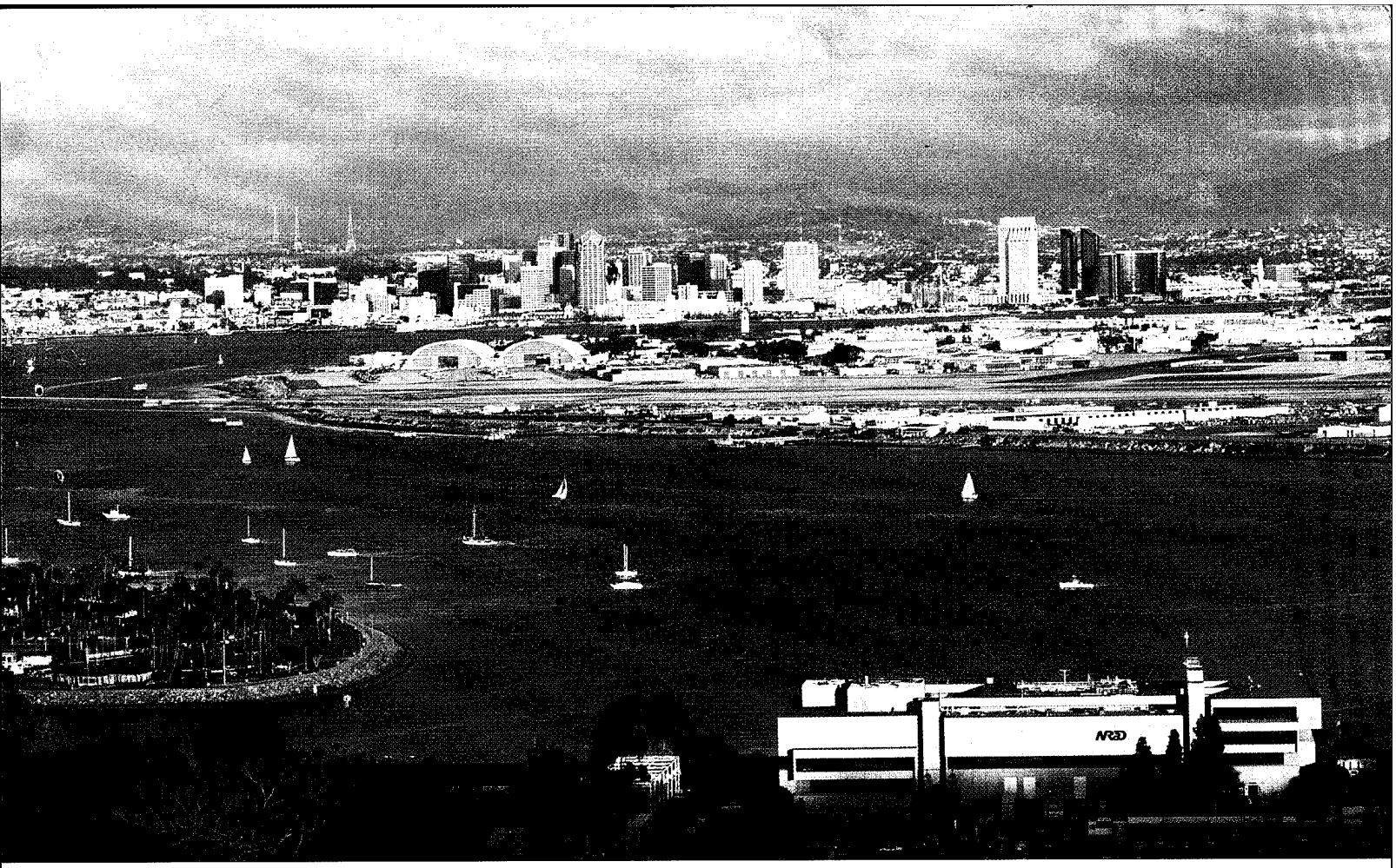


Technical Document 2871
November 1995

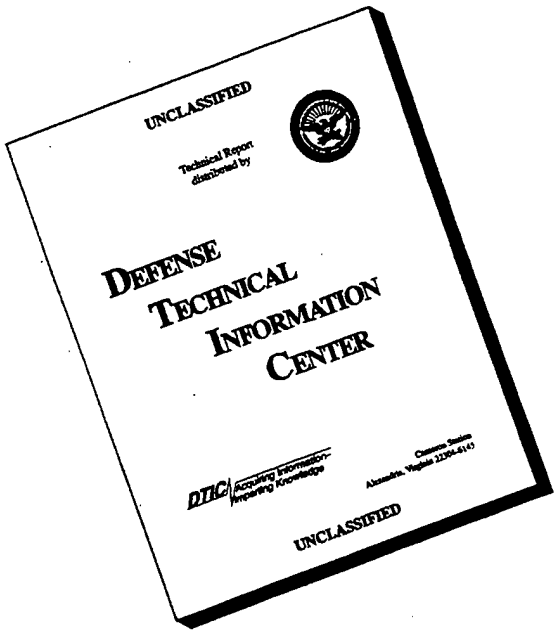
Numerical Electromagnetic Engineering Design System (NEEDS 3.1) Workstation Programmer's Manual

J. C. Lam J. W. Rockway L. C. Russell D. T. Wentworth

Naval Command, Control and
Ocean Surveillance Center
RDT&E Division

San Diego, CA
92152-5001

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Technical Document 2871
November 1995

Numerical Electromagnetic Engineering Design System (NEEDS 3.1) Workstation Programmer's Manual

J. C. Lam
J. W. Rockway
L. C. Russell
D. T. Wentworth

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152-5001

K. E. EVANS, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

The work detailed in this report was performed for the Naval Sea Systems Command by the Naval Command, Control and Ocean Surveillance Center, RDT&E Division, Applied Electromagnetics Branch, Code 851. Funding was provided under Program Element 0603563N.

Released by
L. C. Russell, Head
Applied Electromagnetics Branch

Under authority of
G. W. Beaghler, Head
Electromagnetics Advanced Technology Division

SB

CONTENTS

Table

1. INTRODUCTION

This manual is a programmer's guide to the Numerical Electromagnetic Engineering Design System (NEEDS) Version 3.1 software program. NEEDS was developed to assist users of the Numerical Electromagnetics Code – Method of Moments (NEC-MoM). NEC-MoM requires rigidly defined inputs and often generates massive quantities of output. NEEDS makes NEC-MoM less tedious and more error-free. NEEDS 3.1 is specifically designed for NEC-MoM Version 4 users.

This manual is not required reading for users of NEEDS 3.1. It is designed to provide programmers with insight into how NEEDS 3.1 was developed and implemented on a UNIX workstation. NEEDS was specifically tailored to the Intergraph Workstation and its CLiX operating system. This manual can assist programmers developing similar user interfaces to sophisticated computational codes. This manual details the programming environment used for NEEDS as well as the installation procedures. The appendix contains source code listings for all the major modules used in NEEDS.

Consult the NEEDS Version 3.1 User's Manual (Lam, Rockway, Russell, and Wentworth, 1995) for specific guidance on how to run NEEDS. Refer to Burke (1992) on how to use NEC-MoM Version 4.

Throughout this document, the word NEEDS refers specifically to NEEDS 3.1.

2. PROGRAMMING ENVIRONMENT

NEEDS was written entirely in the C programming language. The previous version of NEEDS (3.0) had several accessory programs that performed filtering of the NEC input and output data sets. These programs had been written in FORTRAN since they predated NEEDS development. In NEEDS 3.1, these accessory programs have been rewritten in C and merged into NEEDS.

2.1 COMMON USER INTERFACE (CUI)

Since NEEDS was developed as part of the Naval Sea Systems Command (NAVSEA) ElectroMagnetic Engineering (EMENG) system, it was required to conform to the EMENG CUI. This requirement ensures a common look and feel for all EMENG software users. The software must run on NAVSEA's CAD 2 (Intergraph) workstation and must be implemented using the X Window System environment and the Motif toolkit. This allows NEEDS to be easily ported to other UNIX-based workstations. The EMENG CUI is documented in Volume VI of Rockwell (1994a).

NAVSEA EM Engineering requirements also specify that all software must be developed using freely available tools. In other words, commercial software packages could not be used since all source code must be freely shared. This created a dilemma in developing the graphics part of the NEEDS package. The Simple Raster Graphics Package (SRGP) and Rockwell's XGraphics library provided a solution.

2.2 SRGP

The SRGP library is freely available over the Internet. It was developed at Brown University (Foley, van Dam, Feiner, and Hughes, 1990). SRGP is a library of functions for creating 2-D engineering graphs. It can be used independently of other software.

Workstations running UNIX and X11 must meet the following requirements to compile SRGP:

1. X11 release 4
2. American National Standards Institute (ANSI) C compiler (GNU's gcc is recommended but not required)
3. Either 4.3 Berkeley Standard Distribution (BSD) or System V UNIX

The software is available from an Internet ftp site. For more information, prepare an e-mail with the words "Software-Distribution" in the Subject line and send it to:

"graphtext@cs.brown.edu"

Software modifications allow the mixing of SRGP routines with Motif. This allows Motif to maintain control of the focus instead of SRGP. Programmers can incorporate the drawing routines into NEEDS instead of using a stand-alone program.

2.3 XGRAPHICS

XGraphics is a library of 3-D drawing functions written for NAVSEA. The XGraphics library contains routines that simplify the creation of graphical images by the application programmer. XGraphics is designed to work in the X Window environment. It is loosely based on the Graphical Kernal System (GKS) 3-D standard.

During NEEDS 3.1 development, XGraphics did not provide a 2-D engineering graphing capability; therefore, it was used solely within NEEDS to provide 3-D visualizations. SRGP was used to provide 2-D engineering graphs for NEEDS.

Information on how to obtain XGraphics may be obtained from the Naval Sea Systems Command (Code 03K24).

2.4 MOSAIC

NEEDS is a complex program with hundreds of input boxes. As a result, it required a fairly sophisticated Help facility. This Help facility was developed using Mosaic. Mosaic is a program for interfacing with the World Wide Web (WWW) of information servers. Its hypertext capability allows users to quickly jump through information and files. It is an excellent, freely available tool for developing hypertext Help. Mosaic files have the extension ".html", which stands for hypertext markup language. There are many tools available for developing ".html" files.

Mosaic source code can be obtained from:

1. "ftp ncsa.uiuc.edu" in the directory "/Mosaic", or
2. "sunsite.unc.edu" in the directory "/pub/packages/inforsystems/WWW".

Compressed binaries are available for many different UNIX platforms. The compression is done using "gzip", which is freely available on the Internet. The files "gzip" and "gunzip" can be obtained from "prep.ai.mit.edu" in "/pub/gnu/gzip-*". (The "*" refers to the most recent release/version). If this site is no longer available, use one of the Internet search engines to find these files at another ftp site.

3. INSTALLATION PROCEDURES

NEEDS 3.1 was designed to run on an Intergraph workstation with a C400 processor under the UNIX operating system and the X Window System. NEEDS 3.1 also requires the installation of the following shared libraries:

1. libXm_s.a
2. libXt_s.a
3. libX11_s.a
4. libc_s.a

These files are normally found in the "/usr/lib" directory. At least 20 MB of disk space should be allowed for the NEEDS 3.1 executable file and various auxiliary files.

The procedure for installing and compiling NEEDS is fairly straightforward. If compilation of NEEDS and its accessory programs is required, then a C compiler, "cc", must be installed on the system (such as in the directory "/usr/bin"). The Makefiles can be modified to reflect the names of the compilers on the specific system. Starting with the file "needs.tar.Z", follow this procedure:

1. Type **uncompress needs.tar.Z** to uncompress "needs.tar".
2. Type **tar -xvfo needs.tar** to restore the file structure under "/needs3.1/".
3. Type **cd needs3.1/sphigs/srgp** to move to the SRGP directory.
4. Type **make** to compile the SRGP library.
5. Type **cd ../../xgraphics/xgclib** to move to the XGraphics directory.
6. Type **make** to compile the Xgraphics library.
7. Type **cd ../..** to move to the NEEDS directory.
8. Type **make** to compile NEEDS and create the needs executable file. (Be sure the Makefiles point to the correct locations for the SRGP and XGraphics libraries).

The binary, "mosaic", must be in a directory in the user's path (such as /usr/bin) to access the Help facility. The app-defaults file "NEEDS" must be in the directory "/usr/lib/X11/app-defaults". This file contains the default values for the X resources that are associated with the NEEDS 3.1 program. The initialization file ".Needs" must be in the user's home directory.

The NEEDS executable program is "needs". To execute the NEEDS program, type **needs**. After testing NEEDS, the system manager may want to put it in a common path such as the directory "/usr/bin".

4. NEEDS SOURCE FILES

The needs program consists of many source files linked together. A separate source file exists for each window in NEEDS. The file names are fairly descriptive. For example, the files "cNodeCoord.c" and "fNodeCoord.c" contain the callbacks and procedures (respectively) for the Node Coordinates Window. The NEEDS Makefile is

```
SRGPDIR = ./sphigs/srgp
EMENG = ./xgraphics/includes
XGRAPHICS = ./xgraphics/xgclib.a
```

```
OBJS = control.o widgets.o dialogs.o cFileMenu.o cResultMenu.o cOutputMenu.o \
        cNodeCoord.o cSWire.o cTaperWire.o \
        cCantWire.o cWireArc.o cHelix.o cFrequency.o cIncident.o cLoads.o \
        cTransLine.o c2PortNet.o cInsulate.o cVoltage.o cMomExport.o \
        cMomImport.o cTransform.o cRotation.o \
        cReflection.o cMaxCoupling.o cNearElect.o cNearMag.o \
        cRadiation.o cPrintOption.o cMeshes.o cSurfPatch.o cMultiPatch.o \
        cAddGround.o cUpMedParam.o cGroundParam.o cDiagnostics.o \
        cNeedControl.o cEditGeo.o cVisual.o \
        fNodeCoord.o fSWire.o fLoads.o fTransLine.o f2PortNet.o \
        fInsulate.o fVoltage.o fMomExport.o \
        fTaperWire.o fCantWire.o fWireArc.o fHelix.o fFrequency.o fIncident.o \
        fMomImport.o fTransform.o fRotation.o fReflection.o fMaxCoupling.o \
        fNearElect.o fNearMag.o fRadiation.o fPrintOption.o \
        fMeshes.o fSurfPatch.o fMultiPatch.o fAddGround.o fUpMedParam.o \
        fGroundParam.o fNeedControl.o fEditGeo.o fComments.o callbacks.o \
        fDiagnostics.o fNecExecute.o fHelp.o needsplt.o fVisual.o \
        fDescrip.o needsflt.o geofilt.o necdisp.o \
        spiral.o modify.o all.o status.o checklist.o dxf.o rdlsol.o

# Files dependent on cFileMenu.h
FM      = cFileMenu.o cMomExport.o cMomImport.o checklist.o dxf.o \ fComments.o fDescrip.o

# Files dependent on control.h
CL      = fNodeCoord.o fSWire.o fTaperWire.o fCantWire.o \
        fWireArc.o fHelix.o fFrequency.o fIncident.o fLoads.o \
        fTransLine.o f2PortNet.o fInsulate.o fVoltage.o \
        fMomExport.o fMomImport.o fTransform.o fRotation.o \
        fReflection.o fMaxCoupling.o fNearElect.o fNearMag.o \
        callbacks.o widgets.o cFileMenu.o

INCLUDE = -I$(SRGPDIR)
LDFLAGS = -L$(SRGPDIR)
LDLIBS  = -lsrgp -lXm -lXt -lX11 -lm
LIBS = -lsrgp -lXm_s -lXt_s -lX11_s -lm -lPW -lbsd -lc_s
CFLAGS = -g -Atarg=c400 -D__cpu_c400__
CC      = acc

needs: $(OBJS)
        $(CC) $(CFLAGS) $(OBJS) $(XGRAPHICS) -o needs $(LDFLAGS) $(LIBS)

$(FM): cFileMenu.h
        $(CC) $(CFLAGS) -c $*.c

$(CL): control.h
        $(CC) $(CFLAGS) -c $*.c

control.o: control.c control.h menuData.h
        $(CC) $(CFLAGS) -c control.c

needsplt.o: tplx11.h
        $(CC) $(CFLAGS) $(INCLUDE) -c needsplt.c $(LDFLAGS) $(LDLIBS)
```

necdisp.o: $(CC)
$(CFLAGS) -DCLIX -DCOMMONS2 -I$(EMENG) -c necdisp.c

Most NEEDS source files are used to create the various NEEDS input/display windows. Almost all of these source files are similar to the source files used to create the Node Coordinates Window. Table 1 lists and describes the files for the Node Coordinates Window. Several other files have unique algorithms and are also included. Listings for these source files are found in appendix A.

Table 1. NEEDS source files.

Filename	Description
control.c control.h	Contain the main control files for NEEDS
menuData.h	Contain the menu data for the NEEDS program
widgets.c	Contain the support procedures for building the widgets
dialogs.c	Contain the general routines for general dialog boxes
cFileMenu.c cFileMenu.h	Contain global variables and Main Menu callbacks for the NEEDS program
cNodeCoord.c fNodeCoord.c	Contain the callbacks and procedures for the Node Coordinates Window
spiral.c	Contain the routines to perform spiral ordering of wires within NEEDS
cMeshes.c fMeshes.c	Contain callbacks and procedures for the Meshes Window
fDescrip.c	Contain the procedures for creating the Description Window
cDiagnostics.c fDiagnostics.c	Contain the callbacks and procedures for the Diagnostics Window
needsplt.c	Contain the SRGP routines for creating the 2-D Engineering Graphics within NEEDS
cOutputMenu.c	Contain the callbacks for the Output menu items
fHelp.c	Contain procedures for creating the Help Window
needs.html needsin1.html	Contain example WWW home pages used by the NEEDS help utility
cVisual.c fVisual.c	Handle selection of 3-D output products
geoflt.c	Extracts NEEDS geometry information from memory
needsflt.c	Extracts output data sets from NEC output files
necdisp.c	Contain procedures for performing 3-D visualizations using XGraphics
cEditGeo.c fEditGeo.c	Contain callbacks and procedures for the Edit Geometry Window
cNeedControl.c fNeedControl.c	Contain callbacks and procedures for the Edit Control Cards Window

5. NEEDS FILES

The only files used by NEEDS are the same as those used by NEC-MoM. This includes the input files to NEC-MoM that have the extension ".nec" and the output files created by NEC-MoM that have the extension "*.out". In addition, there is a set of filtered output files that are extracted from the "*.out" files. These files are documented in the corresponding NEEDS User's Manual (Lam, Rockway, Russell, and Wentworth, 1995).

6. REFERENCES

Burke, G. J. 1992. "Numerical Electromagnetics Code - NEC - 4 Method of Moments Part 1: User's Manual," UCRL-MA-109338 (Jan). Lawrence Livermore National Laboratory, Livermore, CA.

Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes. 1990. *Computer Graphics: Principles and Practice*. Addison-Wesley, New York, NY.

Lam, J. C., J. W. Rockway, L. C. Russell, and D. T. Wentworth. 1995. "Numerical Electromagnetics Engineering Design System (NEEDS 3.1) Workstation User's Manual," NRaD TD 2870. Naval Command, Control and Ocean Surveillance Center, RDT&E Division, San Diego, CA.

Rockwell International Corporation. 1994. "EM Engineering Common Library Software Documentation, Volume VI - EMENG Common User Interface (CUI) Software Description," Contract N00024-89-C-5648 (June). Prepared for Engineering Directorate, Naval Sea Systems Command, Arlington, VA.

A.1 control.c, control.h

control.c:

```
/*
 * User Interface for NEEDS
 */

#include "control.h"
#include "menuData.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Xm/FileSB.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/MainW.h>
#include <Xm/Protocols.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/MwmUtil.h>
#include "ctrlgeo.h"


Widget topLevel, inputFilenameText, necInputFileText, necOutputFileText;
char *necInputFilename  = NULL,
    *necOutputFilename  = NULL;
XtAppContext app_context;

Widget necInputDialog  = NULL,
     necOutputDialog = NULL;

/* Dimension scaling factors */
float DimensionsScale [] = {1, .01, .3048, .0254};

/* Frequency scaling factors */
float FrequenciesScale [] = {.001, 1, 1000};

typedef struct {
  int id;
  char file[40];
} idfile;

/* forward declarations */
static Widget createMenuBar ();
static void   initEnvironment ();
static void   setEnv ();

/* forward declarations of external procs */
extern Widget   createDialogShell ();
extern void     forceUpdate();
extern void     sigcatcher ();
extern idfile   *readIdFile();
extern idfile   *checkInitialStatus();
extern idfile   *IDdata;

/*************************************************************************/
main (argc, argv)
  int argc;
  char *argv[];
{
  Arg args [10];
  Widget mainw, menu_bar;
  int int_val;
  char momfile[80], title[80];
  char *inputname;
  char *tail, newname [80];
  Atom WM_DELETE_WINDOW;
  XEvent event;
  char msg[80];
  extern char *inputFilename;
  extern void exitNeeds ();

  topLevel = XtAppInitialize (&app_context, "NEEDS",
     (XrmOptionDescList)NULL , 0, &argc, (String*)argv, NULL, NULL, 0);

  initEnvironment ();          /* Read the environment variables */
  inputname = getenv("MOM_FILE");
  XtFree(necInputFilename);
  necInputFilename = XtMalloc(strlen(inputname) + 1);
  strcpy(necInputFilename, inputname);
  if (necInputFilename[0] != '\0' && strstr(necInputFilename, ".nec")) {
    strcpy(momfile, necInputFilename);
    readNecFile(inputname);
    strcpy (newname, necInputFilename);
    tail = strstr (newname, ".nec");
    strcpy (tail, ".out");
    XtFree((char *)necOutputFilename);
    necOutputFilename = XtMalloc(strlen(newname) + 1);
    strcpy(necOutputFilename, newname);
```

```
    }
    else {
      if (inputname[0] != '\0' && !strstr(inputname, ".nec")) {
        sprintf(msg, "Improper NEC file extension [%s]", inputname);
        createMessageDialog(topLevel, "Warning", msg, XmDIALOG_WARNING);
        while (True) {
          XtAppNextEvent(app_context, &event);
          XtDispatchEvent(&event);
          if (event.xfocus.type == FocusOut)
            break;
        }
      }
      strcpy (momfile, "New File");
      clearDataInputs();
    }
    sprintf(title, "NEEDS %s - [%s]", VERSION, momfile);

    /* recall nec4s processes which may left over in the last session */
    if (!IDdata)
      IDdata = readIdFile();

    /* Don't allow user to resize this window */
    XtVaGetValues(topLevel, XmNmwmFunctions, &int_val, NULL);
    int_val &= ~(MWM_FUNC_RESIZE | MWM_FUNC_ALL);
    XtVaSetValues(topLevel,
              XmNmwmFunctions, int_val,
              XmNdeleteResponse, XmDO_NOTHING,
                        XmNtitle, title,
              NULL);

    /* Add a callback for the WM_DELETE_WINDOW protocol so that the
     * exitCB callback is called when closing the main window.
     */
    WM_DELETE_WINDOW = XmInternAtom (XtDisplay (topLevel),
                        "WM_DELETE_WINDOW", False);
    XmAddWMProtocolCallback (topLevel, WM_DELETE_WINDOW, exitNeeds, NULL);

    XtSetArg (args [0], XmNheight, 33);
    XtSetArg (args [1], XmNwidth, 800);
    mainw = XtCreateManagedWidget ("mainw", xmMainWindowWidgetClass,
       topLevel, args, 2);

    menu_bar = createMenuBar (mainw);

    XmMainWindowSetAreas (mainw, menu_bar, NULL, NULL, NULL, NULL);
    XtRealizeWidget (topLevel);

    forceUpdate(topLevel);

    /* check to see if any nec4s process is running */
    IDdata = checkInitialStatus(IDdata);

    XtAppMainLoop (app_context);
  } /* end main */

  /*****************************************************************
   * Create the menu bar
   */
  static Widget createMenuBar (parent)
    Widget      parent;
  {
    Widget      menu_bar, menuItem;

    menu_bar = XmCreateMenuBar (parent, "menu_bar", NULL, 0);
    XtManageChild (menu_bar);

    createMenuButtons (NULL, menu_bar, MenuBarData, XtNumber (MenuBarData));

    /* Disable Output */
    menuItem = XtNameToWidget (menu_bar, "Output");
    XtVaSetValues (menuItem, XmNsensitive, False, NULL);

    return (menu_bar);

  } /* end createMenuBar */

  /*****************************************************************
   * initEnvironment
   *
   * Description: Set environment variables using data in file .Needs
   * Added by D.Wentworth 6/13/95
   *****************************************************************/
  static void initEnvironment ()
  {
    char *home, needsDir [80];
    FILE *fp;
    static char *prCommand, *prCommand2, *fontDir, *momFile;
    extern FILE *efopen ();
    char msg[80];
    XEvent event;
    XtAppContext cxt = XtWidgetToApplicationContext(topLevel);

    /* Open .Needs file from user's home directory */
```

```
  needsDir[0] = '\0';
  home = getenv ("HOME");
  if (strcmp(home, "/"))  /* not root login */
    strcpy (needsDir, home);
  strcat (needsDir, "/.Needs");
  if ((fp = efopen (needsDir, "r")) == NULL) {
    while (True) {
      XtAppNextEvent(cxt, &event);
      XtDispatchEvent(&event);
      if (event.xfocus.type == FocusOut)
        break;
    }
    sprintf(msg, "Please create configuration file [%s] then restart",
          needsDir);
    createMessageDialog(topLevel, "Warning", msg, XmDIALOG_WARNING);
    while (True) {
      XtAppNextEvent(cxt, &event);
      XtDispatchEvent(&event);
      if (event.xfocus.type == FocusOut) {
        XtCloseDisplay(XtDisplay(topLevel));
        exit(0);
      }
    }
  }

  setEnv (fp, prCommand);
  setEnv (fp, prCommand2);
  setEnv (fp, fontDir);
  setEnv (fp, momFile);

  fclose (fp);

} /* end initEnvironment */

/*****************************************************************************
 * setEnv
 *
 * Description: Set environment variables using data in file .Needs
 * Added by D.Wentworth 6/13/95
 *****************************************************************************/
static void setEnv (fp, var)
    FILE *fp;
    char *var;
{
  char line [132];

  fgets (line, 132, fp);
  if (line[strlen (line) - 1] == '\n')
    line[strlen(line) - 1] = '\0'; /* Remove carriage return */
  var = XtMalloc (sizeof (char) * (strlen (line) + 1));
  strcpy (var, line);
  putenv(var);

} /* end setEnv */

/*****************************************************************************/
void saveInitEnv(file)
char *file;
{
  char *home, initfile[80], command[80], line[80];
  FILE *fp;

  tmpnam(line);
  home = getenv("HOME");
  if (strcmp(home, "/")) /* not root login */
    sprintf(initfile, "%s/.Needs", home);
  else
    sprintf(initfile, "/.Needs");
  sprintf(command, "head -3 %s > %s", initfile, line);
  system(command);
  sprintf(command, "cat %s > %s", line, initfile);
  system(command);
  remove(line);
  if (!file || !strstr(file, ".nec"))
    sprintf(line, "MOM_FILE=");
  else
    if ((fp = fopen(file, "r")) != NULL) {
      sprintf(line, "MOM_FILE=%s", file);
      fclose(fp);
    }
    else
      sprintf(line, "MOM_FILE=");
  sprintf(command, "echo %s >> %s", line, initfile);
  system(command);
}
```

control.h:

```
/*
 * Header file for user interface of NEEDS-VS
 */
```

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>


typedef struct MenuStruct {
  char                  *name;          /* Name of the button */
  char *accel;                          /* accelerator symbol */
  char *accelText;                      /* accelerator string */
  void                  (*func) ();     /* Callback to be invoked */
  XtPointer    data;          /* Data for the callback */
  struct MenuStruct *subMenu; /* Data for submenu */
  int                   numSubItems;        /* Items in subMenu */
  char                  *subMenuTitle;    /* Title of subMenu */
} XsMenuStruct;

#ifndef FREE_SPACE
/* Options for Environment */

#define FREE_SPACE    0
#define GROUND_PLANE1
#define NUM_ENV                 2       /* Number of Environment options */

/* Options for Dimension */

#define METERS                  0
#define CENTIMETERS   1
#define FEET                    2
#define INCHES                  3
#define NUM_DIM                 4       /* Number of Dimension options */

/* Options for Frequency Units */

#define KHZ             0
#define MHZ             1
#define GHZ             2
#define NUM_FREQ        3       /* Number of Frequency Units options */

#define MAX_COMMANDS            2000    /* Maximum number of commands */

#define NUMSEGS    100
#define NODES      101
#define RADIUS     102

#endif

#define VERSION "3.1"
#define RELEASE "Release Date:  1 AUG 1995"
```

A.2 menuData.h

menuData.h:

```
/*
 * menu data for the NEEDS program
 */

/* forward declarations for callbacks */
extern void clearDataInputs ();
extern void exitNeeds ();
extern void openAboutWindow ();
extern void openAddGroundParamWindow ();
extern void openCatenaryWiresWindow ();
extern void openCommentsWindow ();
extern void openDescripWindow ();
extern void openDiagnosticsWindow ();
extern void openFrequencyWindow ();
extern void openGroundParamWindow ();
extern void openHelixOrSpiralWindow ();
extern void openIncidentPlaneWaveWindow ();
extern void openInsulatedWiresWindow ();
extern void openLoadsWindow ();
extern void openMaxCouplingWindow ();
extern void openMeshesWindow ();
extern void openMomExportWindow ();
extern void openMomImportWindow ();
extern void openMosaicWindow ();
extern void openMultiplePatchWindow ();
extern void openNearElectricWindow ();
extern void openNearMagneticWindow ();
extern void openNecMomExecuteWindow ();
extern void openNodeCoordWindow ();
extern void openPlotDialog ();
extern void openPrintOptionsWindow ();
extern void openPrinterWindow ();
extern void openRadiationPatternWindow ();
extern void openReadInputWindow ();
extern void openRotationWindow ();
extern void openReflectionWindow ();
extern void openSaveAsWindow ();
extern void openStraightWiresWindow ();
extern void openSurfacePatchWindow ();
extern void openTaperedWiresWindow ();
extern void openTransmissionLinesWindow ();
extern void openTransformationWindow ();
extern void openTwoPortNetsWindow ();
extern void openUpperMediumParamWindow ();
extern void openViewAdmittanceWindow ();
extern void openViewAllWindow ();
extern void openViewChargesWindow ();
extern void openViewCouplingWindow ();
extern void openViewCurrentsWindow ();
extern void openViewImpedanceWindow ();
extern void openViewNearElectricFldsWindow ();
extern void openViewNearMagneticFldsWindow ();
extern void openViewRadiationWindow ();
extern void openVisualWindow ();
extern void openVoltageSourcesWindow ();
extern void openWireArcWindow ();
extern void outputMenuCouplingCB ();
extern void saveInputToFile ();
extern void spiral ();
extern void closeAll ();
extern void openNecMomRunStatusWindow ();
extern void openDxfFileWindow();
extern void openImportInputWindow();
extern void openEditCtrlWindow();
extern void openEditWindow();
extern void newFileAction();
extern void necExecute();

static XsMenuStruct FileMenu [] = {
  {"Open...", NULL, NULL, openReadInputWindow},
  {"Save", NULL, NULL, saveInputToFile},
  {"Save as...", NULL, NULL, openSaveAsWindow},
  {"New", NULL, NULL, newFileAction},
  {NULL},
  {"Close all", NULL, NULL, closeAll},
  {NULL},
  {"Exit", NULL, NULL, exitNeeds}
};

static XsMenuStruct spiralMenu [] = {
  {"Positive X-axis", NULL, NULL, spiral, "1"},
  {"Negative X-axis", NULL, NULL, spiral, "2"},
  {"Positive Y-axis", NULL, NULL, spiral, "3"},
  {"Negative Y-axis", NULL, NULL, spiral, "4"},
  {"Positive Z-axis", NULL, NULL, spiral, "5"},
  {"Negative Z-axis", NULL, NULL, spiral, "6"}
};
```

```
static XsMenuStruct GeometryMenu [] = {
  {"Node Coordinates", "Ctrl<Key>N", "Ctl+N", openNodeCoordWindow},
  {"Straight Wires", "Ctrl<Key>W", "Ctl+W", openStraightWiresWindow},
  {"Tapered Wires", "Shift<Key>W", "Shf+W", openTaperedWiresWindow},
  {"Catenary Wires", "Ctrl<Key>C", "Ctl+C", openCatenaryWiresWindow},
  {"Wire Arc", "Ctrl<Key>A", "Ctl+A", openWireArcWindow},
  {"Helix or Spiral", NULL, NULL, openHelixOrSpiralWindow},
  {"Meshes", "Ctrl<Key>Z", "Ctl+Z", openMeshesWindow},
  {"Surface Patches", NULL, NULL, openSurfacePatchWindow},
  {"Multiple Patches", NULL, NULL, openMultiplePatchWindow},
  {"Transformations", "Ctrl<Key>T", "Ctl+T", openTransformationWindow},
  {"Rotations", "Shift<Key>T", "Shf+T", openRotationWindow},
  {"Reflections", "Ctrl<Key>R", "Ctl+R", openReflectionWindow},
  {"Spiral Ordering", NULL, NULL, NULL, NULL, spiralMenu, XtNumber (spiralMenu),
    "Spiral along..."},
  {"CAD Interface", NULL, NULL, openDxfFileWindow},
  {"Edit Card Order", NULL, NULL, openEditWindow}
};

static XsMenuStruct ElectricalMenu [] = {
  {"Frequency", "Ctrl<Key>F", "Ctl+F", openFrequencyWindow},
  {"Loads", "Ctrl<Key>L", "Ctl+L", openLoadsWindow},
  {"Voltage Sources", "Ctrl<Key>V", "Ctl+V", openVoltageSourcesWindow},
  {"Incident Plane Wave", "Ctrl<Key>I", "Ctl+I", openIncidentPlaneWaveWindow},
  {"Transmission Lines", "Alt<Key>L", "Alt+L", openTransmissionLinesWindow},
  {"Two Port Networks", "Ctrl<Key>2", "Ctl+2", openTwoPortNetsWindow},
  {"Insulated Wire", "Shift<Key>L", "Shf+L", openInsulatedWiresWindow},
  {"Ground Parameters", "Ctrl<Key>G", "Ctl+G", openGroundParamWindow},
  {"Additional Ground Parameters", "Alt<Key>G", "Alt+G", openAddGroundParamWindow},
  {"Upper Medium Parameters", "Shift<Key>G", "Shf+G", openUpperMediumParamWindow},
};

static XsMenuStruct SolutionMenu [] = {
  {"Maximum Coupling", "Ctrl<Key>M", "Ctl+M", openMaxCouplingWindow},
  {"Near Electric Fields", "Ctrl<Key>E", "Ctl+E", openNearElectricWindow},
  {"Near Magnetic Fields", "Ctrl<Key>H", "Ctl+H", openNearMagneticWindow},
  {"Radiation Patterns", "Ctrl<Key>R", "Ctl+R", openRadiationPatternWindow},
  {"Print Options", "Ctrl<Key>P", "Ctl+P", openPrintOptionsWindow}
};

static XsMenuStruct translationMenu [] = {
  {"NEC-MoM Export", NULL, NULL, openMomExportWindow},
  {"NEC-MoM Import", NULL, NULL, openImportInputWindow}
};

static XsMenuStruct InputMenu [] = {
  {"Comments", NULL, NULL, openCommentsWindow},
  {"Geometry Description", NULL, NULL, NULL, NULL, GeometryMenu,
    XtNumber (GeometryMenu), NULL},
  {"Edit Control Description", NULL, NULL, openEditCtrlWindow}
};

static XsMenuStruct ExecuteMenu [] = {
  {"Description Summary", "Ctrl<Key>D", "Ctl+D", openDescripWindow},
  {"Diagnostics", "Shift<Key>D", "Shf+D", openDiagnosticsWindow},
  {"NEC-MoM Execute...", "Ctrl<Key>X", "Ctl+X", necExecute},
  {"NEC-MoM Run Status", "Shift<Key>X", "Shf+X", openNecMomRunStatusWindow}
};

static XsMenuStruct textMenu [] = {
  {"All", NULL, NULL, openViewAllWindow},
  {"Impedance", NULL, NULL, openViewImpedanceWindow},
  {"Admittance", NULL, NULL, openViewAdmittanceWindow},
  {"Currents", NULL, NULL, openViewCurrentsWindow},
  {"Charges", NULL, NULL, openViewChargesWindow},
  {"Coupling", NULL, NULL, openViewCouplingWindow},
  {"Near Electric Fields", NULL, NULL, openViewNearElectricFldsWindow},
  {"Near Magnetic Fields", NULL, NULL, openViewNearMagneticFldsWindow},
  {"Radiation Patterns", NULL, NULL, openViewRadiationWindow}
};

static XsMenuStruct plotMenu [] = {
  {"Impedance", NULL, NULL, openPlotDialog, "0"},
  {"Admittance", NULL, NULL, openPlotDialog, "1"},
  {"Currents", NULL, NULL, openPlotDialog, "2"},
  {"Charges", NULL, NULL, openPlotDialog, "3"},
  {"Coupling", NULL, NULL, outputMenuCouplingCB},
  {"Near Electric Fields", NULL, NULL, openPlotDialog, "5"},
  {"Near Magnetic Fields", NULL, NULL, openPlotDialog, "6"},
  {"Radiation Patterns", NULL, NULL, openPlotDialog, "7"},
};

static XsMenuStruct ResultMenu [] = {
  {"Text", NULL, NULL, NULL, NULL, textMenu, XtNumber (textMenu), NULL},
  {"Plot", NULL, NULL, NULL, NULL, plotMenu, XtNumber (plotMenu), NULL},
  {"Visualization...", NULL, NULL, openVisualWindow}
};

static XsMenuStruct OutputMenu [] = {
  {"Printer...", NULL, NULL, openPrinterWindow},
  {"Plotter..."},
  {"Spreadsheet..."},
```

```
  {"Database..."}
};

static XsMenuStruct HelpMenu [] = {
  {"User's Manual", NULL, NULL, openMosaicWindow},
  {NULL},
  {"About NEEDS...", NULL, NULL, openAboutWindow},
};

static XsMenuStruct MenuBarData [] = {
  {"File", NULL, NULL, NULL, NULL, FileMenu, XtNumber (FileMenu), NULL},
  {"Input", NULL, NULL, NULL, NULL, InputMenu, XtNumber (InputMenu), NULL},
  {"Execute", NULL, NULL, NULL, NULL, ExecuteMenu, XtNumber (ExecuteMenu), NULL},
  {"Result", NULL, NULL, NULL, NULL, ResultMenu, XtNumber (ResultMenu), NULL},
  {"Output", NULL, NULL, NULL, NULL, OutputMenu, XtNumber (OutputMenu), NULL},
  {"Help", NULL, NULL, NULL, NULL, HelpMenu, XtNumber (HelpMenu), NULL}
};
```

A.3 widgets.c

widgets.c:

```
/*
 * Support procedures for building the widgets
 */

#include "control.h"
#include <stdio.h>
#include <stdlib.h>
#include <Xm/CascadeB.h>
#include <Xm/DialogS.h>
#include <Xm/Frame.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/List.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>
#include <Xm/ScrollBar.h>
#include <Xm/ScrolledW.h>
#include "actionArea.h"

#define TIGHTNESS 20

extern int      EnvIndex,   /* index into array of Environment options */
                            DimIndex,       /* index into array of Dimension options */
                            FrequencyIndex;   /* index into array of Frequency units */

/*****************************************************************/
Widget createDialogShell (parent, say, w, h)
  Widget   parent;
  char     *say;
  int      w, h;
{

  Widget   widget;
  int      n;
  Arg      args[3];

  n = 0;
  XtSetArg (args [n], XmNwidth, w); n++;
  XtSetArg (args [n], XmNheight, h); n++;
  XtSetArg (args [n], XmNallowShellResize, True);  n++;
  widget = XmCreateDialogShell (parent, say, args, n);

  return (widget);
} /* end createDialogShell */

/*****************************************************************/
void createMenuButtons (title, menu, menuList, numItems)
  char         *title;
  Widget       menu;
  XsMenuStruct  *menuList;
  int          numItems;
{
  Arg          args [10];
  int          i;
  WidgetList    buttons;
  int          separators = 0;
  static       subMenuIndex = 0;  /* number the submenus */

  /* Allocate a widget list to hold all button widgets */
  buttons = (WidgetList) XtMalloc (numItems * sizeof (Widget));

  /* If a title is given, create Label and Separator widgets */
  if (title) {
    XtCreateManagedWidget (title, xmLabelWidgetClass, menu, NULL, 0);
    XtCreateManagedWidget ("separator", xmSeparatorWidgetClass,
       menu, NULL, 0);
  }

  /* Create an entry for each item in the menu */
  for (i = 0; i < numItems; i++) {
    /* A NULL name represents a separator */
    if (menuList[i].name == NULL) {
      XtCreateManagedWidget ("separator", xmSeparatorWidgetClass, menu,
         NULL, 0);
      separators++;
    }

    /* If there is a name and a callback, create a selectable menu entry */
    /* and register the callback function */
    else if (menuList[i].func) {
      Arg nargs[10];
      int n = 0;
      XmString xmStr;
```

```
    xmStr = XmStringCreateLocalized(menuList[i].accelText);
    XtSetArg(nargs[n], XmNaccelerator, menuList[i].accel); n++;
    XtSetArg(nargs[n], XmNacceleratorText, xmStr); n++;
    buttons[i-separators] = XtCreateWidget (menuList[i].name,
      xmPushButtonWidgetClass, menu, nargs, n);
    XmStringFree(xmStr);

    XtAddCallback (buttons[i-separators], XmNactivateCallback,
      menuList[i].func, menuList[i].data);
   }

   /* If there is a name but no callback, the entry must be a label, */
   /* unless there is a submenu. */
   else if (!menuList[i].subMenu) {
    XtSetArg (args [0], XmNalignment, XmALIGNMENT_BEGINNING);
    buttons [i-separators] = XtCreateWidget (menuList[i].name,
      xmLabelWidgetClass, menu, args, 1);
   }

   /* If we got here, the entry must be a submenu.  Create a pulldown */
   /* menu pane and an XmCascadeButton widget.  Attach the menu pane */
   /* and make a recursive call to create the entries in the submenu */

   else {
    Widget   subMenu;
    char     subMenuName[5];

    sprintf (subMenuName, "%d", subMenuIndex++);
/*
    subMenu = XmCreatePulldownMenu (menu, menuList[i].subMenuTitle,
      NULL, 0);
*/
    subMenu = XmCreatePulldownMenu (menu, subMenuName, NULL, 0);
    XtSetArg (args [0], XmNsubMenuId, subMenu);
    buttons [i-separators] = XtCreateWidget (menuList[i].name,
      xmCascadeButtonWidgetClass, menu, args, 1);
    createMenuButtons (menuList[i].subMenuTitle, subMenu,
      menuList[i].subMenu, menuList[i].numSubItems);
   }
  } /* end for */
  XtManageChildren (buttons, numItems - separators);

  XtSetArg (args [0], XmNmenuHelpWidget, buttons [numItems-1]);
  XtSetValues (menu, args, 1);

  XtFree ((char *) buttons);

} /* end createMenuButtons */

/*****************************************************************/

void createRadioBoxItems (box, xmstrings)
  Widget   box;
  XmString *xmstrings;
{
  Widget   button;
  int      i,
           n;
  char     name [2];
  Arg      args [2];

  n = XtNumber (xmstrings);
  for (i = 0; i < n; i++) {
   sprintf (name, "%d", i);
   XtSetArg (args [0], XmNlabelString, xmstrings [i]);
   XtSetArg (args [1], XmNalignment, XmALIGNMENT_BEGINNING);
   button = XmCreateToggleButton (box, name, args, 2);
   XtManageChild (button);
  }
} /* createRadioBoxItems */

/*****************************************************************/

char *getStringFromXmString (string)
  XmString                string;
{
  XmStringContext         context;
  char                    *text;
  XmStringCharSet         charSet;
  XmStringDirection       dir;
  Boolean                 separator;
  char                    *buf = NULL;
  int                     done = FALSE;

  XmStringInitContext (&context, string);
  while (!done)
   if (XmStringGetNextSegment (context, &text, &charSet, &dir, &separator)) {

    if (separator)          /* Stop when next segment is a separator */
     done = TRUE;

    if (buf) {
     buf = XtRealloc (buf, strlen (buf) + strlen (text) + 2);
```

```
            strcat (buf, text);
        } else {
          buf = (char *) XtMalloc (strlen (text) + 1);
          strcpy (buf, text);
        }
        XtFree (text);
      } else
        done = TRUE;

    XmStringFreeContext (context);
    return (buf);
  } /* end getStringFromXmString */

  /*****************************************************************/
  void createOptionMenu (parent, menuData, frame, options)
    Widget    parent,
              *frame,
              *options;
    char      **menuData;
  {
    Widget    menuPane,
              optionMenu;
    int       n,
              i;
    Arg       args [10];
    XmString  string;

    /* Create Option Menu */
    n = 0;
    XtSetArg (args [n], XmNshadowType, XmSHADOW_ETCHED_IN);  n++;
    *frame = XmCreateFrame (parent, "frame", args, n);
    XtManageChild (*frame);

    n = 0;
    menuPane = XmCreatePulldownMenu (*frame, "menuPane", args, n);

    for (i = 1; menuData [i] != NULL; i++) {
      n = 0;
      options [i-1] = XmCreatePushButton (menuPane, menuData [i], args, n);
      XtManageChild (options [i-1]);
    }

    string = XmStringCreateSimple (menuData [0]);
    n = 0;
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetArg (args [n], XmNsubMenuId, menuPane); n++;
    optionMenu = XmCreateOptionMenu (*frame, menuData [0], args, n);
    XtManageChild (optionMenu);
    XtFree ((char *) string);

  } /* end createEnvironDimensOptionMenus */

  /*****************************************************************/
  void saveEnvironment (env)

    Widget  env;
  {
    String  envString;

    envString = XtName (env);
    if (strcmp (envString, "Free Space") == 0)
      EnvIndex = FREE_SPACE;
    else
      EnvIndex = GROUND_PLANE;

  } /* saveEnvironment */

  /*****************************************************************/
  void saveDimension (dim)

    Widget  dim;
  {
    String  dimString;

    dimString = XtName (dim);
    if (strcmp (dimString, "Meters") == 0)
      DimIndex = METERS;
    else if (strcmp (dimString, "Centimeters") == 0)
      DimIndex = CENTIMETERS;
    else if (strcmp (dimString, "Feet") == 0)
      DimIndex = FEET;
    else
      DimIndex = INCHES;

  } /* saveEnvironment */

  /*****************************************************************/
  void saveFrequencyUnits (freqUnits)

    Widget  freqUnits;
  {
    String  freqString;
```

```
  freqString = XtName (freqUnits);
  if (strcmp (freqString, "KHz") == 0)
    FrequencyIndex = KHZ;
  else if (strcmp (freqString, "MHz") == 0)
    FrequencyIndex = MHZ;
  else
    FrequencyIndex = GHZ;

} /* saveFrequencyUnits */

/*****************************************************************/
int getListPosition (listWidget)
  Widget  listWidget;
{
  int    *positionList,
         count = 0,
         nodeIndex;

/* Get the current list selection */

  if (XmListGetSelectedPos (listWidget, &positionList, &count)) {
    nodeIndex = positionList [0];
    XtFree ((char *) positionList);
  } else
    nodeIndex = 0;

  return (nodeIndex);

} /* end getListPosition */

/*****************************************************************/
Widget createActionArea (parent, actions, numActions)

  Widget        parent;
  ActionAreaItem  *actions;
  int           numActions;
{
  Widget        actionArea,
                widget;
  int           n, i;
  Arg           args [3];

  n = 0;
  if (XmIsForm (parent)) {
    XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  }
  XtSetArg (args [n], XmNfractionBase,  TIGHTNESS * numActions - 1); n++;
  actionArea = XmCreateForm (parent, "actionArea", args, n);

  for (i = 0; i < numActions; i++) {
    widget = XtVaCreateManagedWidget (actions[i].label,
      xmPushButtonWidgetClass,  actionArea,
      XmNleftAttachment,        i ? XmATTACH_POSITION : XmATTACH_FORM,
      XmNleftPosition,          TIGHTNESS * i,
      XmNtopAttachment,         XmATTACH_FORM,
      XmNbottomAttachment,      XmATTACH_FORM,
      XmNrightAttachment,
        i != numActions - 1 ? XmATTACH_POSITION : XmATTACH_FORM,
      XmNrightPosition,         TIGHTNESS * i + (TIGHTNESS -1),
      XmNshowAsDefault,         i == 0,
      XmNdefaultButtonShadowThickness, 1,
      NULL);
    if (actions[i].callback)
      XtAddCallback (widget, XmNactivateCallback, actions[i].callback,
        actions[i].data);

    /* Set the actionArea's default button to the first widget created (or,
     * make the index a parameter to the function or have it be part of the
     * data structure).  Also, set the pane window constraint for max and
     * min heights so this pane in the widget is not resizable.
     */
    if (i == 0) {
      Dimension             height, h;

      XtVaGetValues (actionArea, XmNmarginHeight, &h, NULL);
      XtVaGetValues (widget, XmNheight, &height, NULL);
      height += 2 * h;
      XtVaSetValues (actionArea,
        XmNdefaultButton,  widget,
        XmNpaneMaximum,                         height,
        XmNpaneMinimum,           height,
        NULL);
    }
  }
  XtManageChild (actionArea);

  return (actionArea);

} /* end createActionArea */

/*****************************************************************
 * For the input screens, this procedure updates the string which
```

```
 * displays the current list index and count.
 ********************************************************************/
void updateIndexLabel (label, index, count)

  Widget      label;
  int         index,
              count;
{
  char        str[80];
  XmString    string;

  sprintf (str, "Index %d of %d", index, count);
  string = XmStringCreateSimple (str);
  XtVaSetValues (label, XmNlabelString, string, NULL);
  XmStringFree (string);

} /* end updateIndexLabel */

/********************************************************************
 * If "valid" boolean is false, display an error dialog indicating
 * which input needs revision.
 ********************************************************************/

Boolean valueCheck (parent, name, valid)
  Widget      parent;
  char        *name;
  Boolean     valid;
{
  char        format [] = "Invalid %s.  Please reenter.",
              msg [80];

  if (!valid) {
    sprintf (msg, format, name);
    createMessageDialog (parent, "Error", msg, XmDIALOG_ERROR);
    return (False);
  }
  return (True);

} /* end valueCheck */

/********************************************************************
 * Before a node text widget loses focus, make sure that its value is
 * not equal to the other node text widget.
 ********************************************************************/

Boolean nodeCheck (parent, end1, end2)
  int         end1, end2;
{
  extern int   NodeCount;

  if (end1 == end2) {
    createMessageDialog (parent, "Error",
       "Zero length wire.  Please reenter wire end values.", XmDIALOG_ERROR);
    return (False);
  }
  if ((end1 < 1) || (end1 > NodeCount) || (end2 < 1) || (end2 > NodeCount)) {
    createMessageDialog (parent, "Error",
       "Invalid node value.  Please reenter.", XmDIALOG_ERROR);
    return (False);
  }
  return (True);

} /* end nodeCheckCB */

/********************************************************************/
typedef struct {
   Widget text;
   char *data;
   int numdata;
   short index;
   unsigned char type;
} scrollControl;

void decrementCB(w, control, call_data)
Widget w;
scrollControl *control;
XtPointer call_data;
{
  char str[40];

  control->index += 1;
  if (control->index == control->numdata)
    control->index = 0;
  if (control->type) {
    int *data = (int *)control->data;
    sprintf(str, "%d", data[control->index]);
  }
  else {
    float *data = (float *)control->data;
    sprintf(str, "%g", data[control->index]);
  }
  XmTextSetString(control->text, str);
```

```
  call_data = NULL;
  w = NULL;
}

void incrementCB(w, control, call_data)
Widget w;
scrollControl *control;
XtPointer call_data;
{
  char str[40];

  control->index -= 1;
  if (control->index < 0)
    control->index = control->numdata - 1;
  if (control->type) {
    int *data = (int *)control->data;
    sprintf(str, "%d", data[control->index]);
  }
  else {
    float *data = (float *)control->data;
    sprintf(str, "%g", data[control->index]);
  }
  XmTextSetString(control->text, str);

  call_data = NULL;
  w = NULL;
}

void valueChangedCB(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
  Arg args[10];
  int size, max, min, value, n = 0;

  XtSetArg(args[n], XmNsliderSize, &size); n++;
  XtSetArg(args[n], XmNmaximum, &max); n++;
  XtSetArg(args[n], XmNminimum, &min); n++;
  XtGetValues(w, args, n);
  value = (min + (max - size)) / 2;
  XtVaSetValues(w, XmNvalue, value, NULL);

  call_data = NULL;
  client_data = NULL;
}

/*****************************************************************/
Widget createScrolledText(parent, name, arg, numArg, data, numdata, type)
Widget parent;
char *name;
ArgList arg;
int numArg;
char *data;
int numdata;
unsigned char type;
{
  Widget text, scrollBar, scrolledWindow;
  static scrollControl control[10];
  Arg args[10];
  int size, max, min, value, i = 0, n = 0;
  char str[40];
  void incrementCB();
  void decrementCB();
  void valueChangedCB();
  void exposeHandler();

  scrolledWindow = XmCreateScrolledWindow(parent, "win", NULL, 0);
  scrollBar = XmCreateScrollBar(scrolledWindow, "scroll", NULL, 0);
  text = XmCreateText (scrolledWindow, name, arg, numArg);
  if (type)
    sprintf(str, "%d", *(int *)data);
  else
    sprintf(str, "%g", *(float *)data);
  XmTextSetString(text, str);
  XmScrolledWindowSetAreas(scrolledWindow, NULL, scrollBar, text);
  XtManageChild (scrolledWindow);
  XtSetArg(args[n], XmNsliderSize, &size); n++;
  XtSetArg(args[n], XmNmaximum, &max); n++;
  XtSetArg(args[n], XmNminimum, &min); n++;
  XtGetValues(scrollBar, args, n);
  value = (min + (max - size)) / 2;
  XtVaSetValues(scrollBar, XmNvalue, value, NULL);
  while (control[i].data) {
    if (control[i].data == data)
      break;
    i++;
  }
  control[i].text = text;
  control[i].data = data;
  control[i].numdata = numdata;
  control[i].index = 0;
  control[i].type = type;
```

```
  XtAddCallback(scrollBar, XmNincrementCallback, incrementCB, &control[i]);
  XtAddCallback(scrollBar, XmNdecrementCallback, decrementCB, &control[i]);
  XtAddCallback(scrollBar, XmNvalueChangedCallback, valueChangedCB, NULL);
  XtAddEventHandler(text, StructureNotifyMask, False,
            exposeHandler, scrollBar);

  return text;
}

/*****************************************************************/
void editScrolledText (w, data, numdata, type)
Widget w;
char *data;
int numdata;
unsigned char type;
{
 Widget scrollBar;
 static scrollControl control[10];
 int i = 0;
 char str[40];
 void incrementCB();
 void decrementCB();

 if (type)
  sprintf(str, "%d", *(int *)data);
 else
  sprintf(str, "%g", *(float *)data);
 XmTextSetString(w, str);
 while (control[i].data) {
  if (control[i].data == data)
   break;
  i++;
 }
 control[i].text = w;
 control[i].data = data;
 control[i].numdata = numdata;
 control[i].index = 0;
 control[i].type = type;
 XtVaGetValues(XtParent(w), XmNverticalScrollBar, &scrollBar, NULL);
 XtRemoveAllCallbacks(scrollBar, XmNincrementCallback);
 XtRemoveAllCallbacks(scrollBar, XmNdecrementCallback);
 XtAddCallback(scrollBar, XmNincrementCallback, incrementCB, &control[i]);
 XtAddCallback(scrollBar, XmNdecrementCallback, decrementCB, &control[i]);
}

/*****************************************************************/
void exposeHandler (w, bar, event)
Widget w, bar;
XEvent *event;
{
 if (event->xany.type == MapNotify)
  XtManageChild(bar);
 if (event->xany.type == UnmapNotify)
  XtUnmanageChild(bar);

 w = NULL;
}
```

A.4 dialogs.c

dialogs.c:

```
/*
 * Filename: dialogs.c
 *
 * contains general routines for general dialog boxes such as
 * the FileSelection & Error dialogs
 */

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/FileSB.h>
#include <Xm/MessageB.h>
#include <Xm/SelectioB.h>
#include <string.h>
#include <Xm/PushB.h>
#include <Xm/Text.h>
#include <Xm/Label.h>
#include <Xm/MwmUtil.h>
#include "control.h"

Boolean SkipPrompt = False;
extern Widget  topLevel;

extern void forceUpdate();
extern Widget getTopShell();

/*****************************************************************/
void dialogCancelCB(w, data, cbs)
Widget w;
XtPointer data;
XmAnyCallbackStruct *cbs;
{
 XtDestroyWidget(XtParent(w));

 data = NULL;
 cbs = NULL;

} /* end dialogCancelCB */

/*****************************************************************/
Widget createFileSelectionDialog (title, filter)
 char    *title,
         *filter;
{
 Arg     args [2];
 XmString   xmFilter,
         xmTitle;
 int     n;
 Widget  fileSelectionDialog;

 xmFilter = XmStringLtoRCreate(filter, XmSTRING_DEFAULT_CHARSET);
 xmTitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);

 n = 0;
 XtSetArg (args [n], XmNdirMask, xmFilter); n++;
 XtSetArg (args [n], XmNdialogTitle, xmTitle); n++;
 fileSelectionDialog = XmCreateFileSelectionDialog
   (topLevel, "dialog", args, n);
 XmStringFree (xmFilter);
 XmStringFree (xmTitle);

 XtUnmanageChild (XmFileSelectionBoxGetChild (fileSelectionDialog,
                  XmDIALOG_HELP_BUTTON));
 XtAddCallback (fileSelectionDialog, XmNcancelCallback,
          (XtCallbackProc) XtUnmanageChild, NULL);

 return (fileSelectionDialog);

} /* end createFileSelectionDialog */

/*****************************************************************
 * Opens a Message dialog box.
 */
void createMessageDialog (parent, title, message, type)
 Widget      parent;
 char        *title,
             *message;
 int         type;
{
 Widget      messageBox;
 XmString    string,
             okString,
             xmtitle;
 Arg         args [5];
 int         n = 0;

 string = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
```

```
    okString = XmStringCreateSimple ("OK");
    xmtitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
    XtSetArg (args [n], XmNmessageString, string); n++;
    XtSetArg (args [n], XmNautoUnmanage, False); n++;
    XtSetArg (args [n], XmNcancelLabelString, okString); n++;
    XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
    messageBox = XmCreateMessageDialog (parent, "Message", args, n);

    XtAddCallback (messageBox, XmNcancelCallback,
                (XtCallbackProc) dialogCancelCB, NULL);

    XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_OK_BUTTON));
    XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));

    XtVaSetValues (messageBox, XmNdialogType,
                type == NULL ? XmDIALOG_MESSAGE : type, NULL);

    XmStringFree (string);
    XmStringFree (okString);
    XmStringFree (xmtitle);

    XtVaSetValues(getTopShell(messageBox), XmNmwmInputMode,
                MWM_INPUT_SYSTEM_MODAL, NULL);

    XBell (XtDisplay (messageBox), 50);
    XtManageChild (messageBox);
    forceUpdate (messageBox);

} /* end createMessageDialog */

/*****************************************************************************
 * Opens a Prompt dialog box.
 */
void createPromptDialog (prompt, title, callback)
    char        *prompt,
                *title;
    void        (*callback) ();
{
    Widget      dialog,
                child;
    Arg         args [6];
    int                         n = 0;
    XmString    str,
                xmtitle;

    str = XmStringCreateSimple (prompt);
    xmtitle = XmStringCreateSimple (title);
    XtSetArg (args[n], XmNselectionLabelString, str); n++;
    XtSetArg (args[n], XmNautoUnmanage, False); n++;
    XtSetArg (args[n], XmNdialogTitle, xmtitle); n++;
    XtSetArg (args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
    dialog = XmCreatePromptDialog (topLevel, "prompt", args, n);
    XmStringFree (str);
    XmStringFree (xmtitle);

    child = XmSelectionBoxGetChild (dialog, XmDIALOG_TEXT);
    XtAddCallback (dialog, XmNokCallback, callback, (XtPointer) child);

    /* Destroy the dialog when user selects cancel */
    XtAddCallback (dialog, XmNcancelCallback, (XtCallbackProc) XtDestroyWidget,
                NULL);

    /* Remove the Help button */
    child = XmSelectionBoxGetChild (dialog, XmDIALOG_HELP_BUTTON);
    XtUnmanageChild (child);

    XtVaSetValues(getTopShell(dialog), XmNmwmInputMode,
                MWM_INPUT_SYSTEM_MODAL, NULL);

    XtManageChild (dialog);

} /* end createPromptDialog */

/*****************************************************************************
 * Returns the filename from the FileSelectionDialog.
 */
char *getFilename (callbackStruct)
    XmFileSelectionBoxCallbackStruct    *callbackStruct;
{
    char                        *filename;

    if (!XmStringGetLtoR (callbackStruct->value, XmSTRING_DEFAULT_CHARSET,
        &filename)) {
        return (NULL);          /* Must have been an internal error */
    }
    if (filename[strlen(filename) - 1] == '/') {
        createMessageDialog (topLevel, "Error", "No file selected.",
                        XmDIALOG_MESSAGE);
        XtFree (filename);      /* even "" is an allocated byte */
        return (NULL);
    }
    if (!*filename) {           /* Nothing typed? */
```

```
    createMessageDialog (topLevel, "Error", "No file selected.",
                XmDIALOG_MESSAGE);
    XtFree (filename);      /* even "" is an allocated byte */
    return (NULL);
   }
   return (filename);

} /* end getFilename */

/*****************************************************************
 * Try to open a file.  If it's not possible, open a Message Dialog
 * box warning the user.
 */

FILE *efopen (file, mode)
  char                  *file,
                        *mode;
{
  FILE                  *fp;
  char                  msg [80];

  if ((fp = fopen (file, mode)) != NULL)
    return (fp);
  sprintf (msg, "Can't open file %s", file);
  createMessageDialog (topLevel, "Error", msg, XmDIALOG_ERROR);
  return (NULL);

} /* end efopen */

/*****************************************************************
 * Opens a Message dialog box.
 */
void changeFocus(w, focus, call_data)
Widget w, focus;
XmAnyCallbackStruct *call_data;
{
  Widget shell = XtParent(w);
  Display *dpy = XtDisplay(w);
  Window win = XtWindow(focus);
  Widget bfocus;
  XWindowAttributes att;
  long mask;
  extern Widget getTopShell();

  bfocus = XmGetFocusWidget(getTopShell(focus));
  if (XmIsPushButton(bfocus))
    XtCallActionProc(bfocus, "Disarm", NULL, NULL, 0);
  XtSetKeyboardFocus(getTopShell(focus), focus);
  XGetWindowAttributes(dpy, win, &att);
  mask = att.your_event_mask ^ ButtonPressMask;
  XSelectInput(dpy, win, mask);
  XtDestroyWidget(shell);

  call_data = NULL;

} /* end changeFocus */

/*****/
void continueFocus(w, focus, call_data)
Widget w, focus;
XmAnyCallbackStruct *call_data;
{
  Widget shell = XtParent(w);
  Widget bfocus;
  extern Widget getTopShell();

  bfocus = XmGetFocusWidget(getTopShell(focus));
  if (XmIsPushButton(bfocus))
    XtCallActionProc(bfocus, "Disarm", NULL, NULL, 0);
  XtDestroyWidget(shell);

  call_data = NULL;

} /* end continueFocus */

/*****/
void createMessageDialog2 (parent, title, message, type, focus)
  Widget        parent;
  char          *title,
                *message;
  int           type;
  Widget        focus;
{
  Widget        messageBox;
  XmString      string,
                okString,
                xmtitle;
  Arg           args [5];
  int           n = 0;

  string = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
  okString = XmStringCreateSimple ("OK");
  xmtitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);
```

```
    XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
    XtSetArg (args [n], XmNmessageString, string); n++;
    XtSetArg (args [n], XmNautoUnmanage, False); n++;
    XtSetArg (args [n], XmNcancelLabelString, okString); n++;
    XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
    messageBox = XmCreateMessageDialog (parent, "Message", args, n);

    if (type == XmDIALOG_ERROR)
      XtAddCallback (messageBox, XmNcancelCallback,
                (XtCallbackProc) changeFocus, focus);
    else
      XtAddCallback (messageBox, XmNcancelCallback,
                (XtCallbackProc) continueFocus, focus);

    XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_OK_BUTTON));
    XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));

    XtVaSetValues (messageBox, XmNdialogType,
              type == NULL ? XmDIALOG_MESSAGE : type, NULL);

    XmStringFree (string);
    XmStringFree (okString);
    XmStringFree (xmtitle);

    XtVaSetValues(getTopShell(messageBox), XmNmwmInputMode,
              MWM_INPUT_SYSTEM_MODAL, NULL);

    XBell (XtDisplay (messageBox), 50);
    XtManageChild (messageBox);

  } /* end createMessageDialog2 */

  /*************************************************************************
   * Opens a Exit dialog box.
   */
  static Boolean textValidCheck (text)
  Widget text;
  {
    char *filename;
    char *tail, msg[80];
    Display *dpy = XtDisplay(text);
    Window win = XtWindow(text);
    XWindowAttributes att;
    long mask;
    extern char *necInputFilename;
    extern void createMessageDialog2();
    extern Widget getTopShell();

    XGetWindowAttributes(dpy, win, &att);
    mask = att.your_event_mask | ButtonPressMask;
    XSelectInput(dpy, win, mask);

    filename = XmTextGetString(text);

    if (necInputFilename)
      if (strcmp(filename, necInputFilename) == 0)
        return True;

    if (!filename[0]) {
      XtFree (necInputFilename);
      XtFree (filename);
      necInputFilename = NULL;
      sprintf(msg, "Empty file name");
      createMessageDialog2(getTopShell(text), "Error", msg,
                  XmDIALOG_ERROR, text);
      XtFree (filename);
      return False;
    }

    tail = strstr (filename, ".nec");
    if (!tail) {
      sprintf(msg, "Improper file extension %s", filename);
      createMessageDialog2(getTopShell(text), "Error", msg,
                  XmDIALOG_ERROR, text);
      XtFree (filename);
      return False;
    }
    XtFree (necInputFilename);
    necInputFilename = XtMalloc(strlen(filename) + 1);
    strcpy(necInputFilename, filename);
    XtFree (filename);
    return True;

  } /* end textValidCheck */

  /*****/
  void exitOkCallback(w, text, call_data)
  Widget w, text;
  XmAnyCallbackStruct *call_data;
  {
    Widget shell = XtParent(w);
    extern Boolean sphigsOff;
    extern char *necInputFilename;
```

```
  extern char *inputFilename;
  extern void momExportAction();
  extern void saveInitEnv();

  if (!textValidCheck(text))
    return;

  if (necInputFilename && necInputFilename[0]) {
    momExportAction(necInputFilename);
  }

  saveInitEnv(necInputFilename);   /* Save current *.nec into init file */
  if (!sphigsOff) SRGP_end (); /* Disable SRGP */

  XtDestroyWidget(shell);
  XtCloseDisplay (XtDisplay(w));

  call_data = NULL;
  exit (0);

} /* end exitOkCallback */

void exitCancelCallback(w, client_data, call_data)
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *call_data;
{
  extern char *necInputFilename;
  extern void saveInitEnv();
  extern Boolean sphigsOff;

  saveInitEnv(necInputFilename);   /* Save current *.nec into init file */
  if (!sphigsOff) SRGP_end (); /* Disable SRGP */

  XtCloseDisplay (XtDisplay(w));

  client_data = NULL;
  call_data = NULL;
  exit (0);

} /* end exitCancelCallback */

void createExitDialog (parent, title, message)
 Widget       parent;
 char         *title,
              *message;
{
 Widget       messageBox, text;
 XmString     string,
              okString,
              cancelString,
              xmtitle;
 Arg          args [10];
 int          n = 0;
 extern char *necInputFilename;

 string = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
 okString = XmStringCreateSimple ("OK");
 cancelString = XmStringCreateSimple ("Cancel");
 xmtitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);
 XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
 XtSetArg (args [n], XmNmessageString, string); n++;
 XtSetArg (args [n], XmNautoUnmanage, False); n++;
 XtSetArg (args [n], XmNcancelLabelString, cancelString); n++;
 XtSetArg (args [n], XmNokLabelString, okString); n++;
 XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
 messageBox = XmCreateMessageDialog (parent, "Message", args, n);

 text = XtCreateManagedWidget("text", xmTextWidgetClass,
                messageBox, NULL, 0);
 if (necInputFilename && necInputFilename[0])
   XmTextSetString(text, necInputFilename);
 else
   XmTextSetString(text, "untitled.nec");

 XtAddCallback (messageBox, XmNokCallback,
          (XtCallbackProc) exitOkCallback, text);
 XtAddCallback (messageBox, XmNcancelCallback,
          (XtCallbackProc) exitCancelCallback, NULL);
 XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));

 XtVaSetValues (messageBox, XmNdialogType, XmDIALOG_WARNING, NULL);

 XmStringFree (string);
 XmStringFree (okString);
 XmStringFree (cancelString);
 XmStringFree (xmtitle);

 XtVaSetValues(getTopShell(messageBox), XmNmwmInputMode,
          MWM_INPUT_SYSTEM_MODAL, NULL);

 XBell (XtDisplay (messageBox), 50);
 XtManageChild (messageBox);
```

```
} /* end createExitDialog */

/*****************************************************************/
void promptMessageOkCB(w, text, cbs)
Widget w;
Widget text;
XmAnyCallbackStruct *cbs;
{
 extern void openMomImportWindow();
 extern char *momFilename;

 XtFree (momFilename);
 momFilename = XmTextGetString(text);

 openMomImportWindow();
 XtDestroyWidget(XtParent(w));

 cbs = NULL;

} /* end promptMessageOkCB */

/*****************************************************************/
void promptMessageOkCB2(w, text, cbs)
Widget w;
Widget text;
XmAnyCallbackStruct *cbs;
{
 extern char *exportFilename;
 extern char *necInputFilename;
 extern void momExportAction();

 XtFree (exportFilename);
 exportFilename = XmTextGetString(text);
 momExportAction(exportFilename);
 XtFree (necInputFilename);
 necInputFilename = strdup(exportFilename);
 XtDestroyWidget(XtParent(w));

 cbs = NULL;

} /* end promptMessageOkCB2 */

/*****************************************************************/
void promptMessageCancelCB(w, data, cbs)
Widget w;
XtPointer data;
XmAnyCallbackStruct *cbs;
{
 SkipPrompt = True;
 XtDestroyWidget(XtParent(w));

 data = NULL;
 cbs = NULL;

} /* end promptMessageCancelCB */

/*****************************************************************/
void createPromptMessageDialog(parent, title, message, type, file)
 Widget      parent;
 char        *title,
             *message;
 int         type;
 char *file;
{
 Widget      messageBox, text;
 XmString    string,
             xmtitle;
 Arg         args [5];
 int         n = 0;
 char *ext;

 string = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
 xmtitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);
 XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
 XtSetArg (args [n], XmNmessageString, string); n++;
 XtSetArg (args [n], XmNautoUnmanage, False); n++;
 XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
 messageBox = XmCreateMessageDialog (parent, "Message", args, n);
 XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));
 XtVaSetValues (messageBox, XmNdialogType,
         type == NULL ? XmDIALOG_MESSAGE : type, NULL);
 XmStringFree (string);
 XmStringFree (xmtitle);

 XtSetArg (args [0], XmNeditMode, XmSINGLE_LINE_EDIT);
 XtSetArg (args [1], XmNcolumns, 30);
 text = XmCreateText (messageBox, "text", args, 2);
 XtManageChild (text);
 XmTextSetString(text, file);
 XmTextSetInsertionPosition(text, strlen(file));
 XtVaSetValues(messageBox, XmNinitialFocus, text, NULL);
 XtAddCallback (messageBox, XmNcancelCallback,
```

```
            (XtCallbackProc) promptMessageCancelCB, NULL);
  ext = strrchr(file, '.');
  if (strcmp(ext, ".mom") == 0)
    XtAddCallback (messageBox, XmNokCallback,
            (XtCallbackProc) promptMessageOkCB, text);
  else if (strcmp(ext, ".nec") == 0)
    XtAddCallback (messageBox, XmNokCallback,
            (XtCallbackProc) promptMessageOkCB2, text);

  XtVaSetValues(getTopShell(messageBox), XmNmwmInputMode,
            MWM_INPUT_SYSTEM_MODAL, NULL);

  XBell (XtDisplay (messageBox), 50);
  XtManageChild (messageBox);

} /* end createPromptMessageDialog */

/*****************************************************************/
void createPromptMessageDialog2(parent, title, message, type, file)
  Widget       parent;
  char         *title,
               *message;
  int          type;
  char *file;
{
  Widget       messageBox, text;
  XmString     string, okString, cancelString,
               xmtitle;
  Arg          args [10];
  int          n = 0;
  char *ext;

  string = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
  xmtitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);
  XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
  XtSetArg (args [n], XmNmessageString, string); n++;
  XtSetArg (args [n], XmNautoUnmanage, False); n++;
  XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
  okString = XmStringCreateSimple ("Continue");
  cancelString = XmStringCreateSimple ("Skip");
  XtSetArg (args [n], XmNcancelLabelString, cancelString); n++;
  XtSetArg (args [n], XmNokLabelString, okString); n++;
  messageBox = XmCreateMessageDialog (parent, "Message", args, n);
  XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));
  XtVaSetValues (messageBox, XmNdialogType,
            type == NULL ? XmDIALOG_MESSAGE : type, NULL);
  XmStringFree (string);
  XmStringFree (xmtitle);
  XmStringFree (okString);
  XmStringFree (cancelString);

  XtSetArg (args [0], XmNeditMode, XmSINGLE_LINE_EDIT);
  XtSetArg (args [1], XmNcolumns, 30);
  text = XmCreateText (messageBox, "text", args, 2);
  XtManageChild (text);
  XmTextSetString(text, file);
  XmTextSetInsertionPosition(text, strlen(file));
  XtVaSetValues(messageBox, XmNinitialFocus, text, NULL);
  XtAddCallback (messageBox, XmNcancelCallback,
            (XtCallbackProc) promptMessageCancelCB, NULL);
  ext = strrchr(file, '.');
  if (strcmp(ext, ".mom") == 0)
    XtAddCallback (messageBox, XmNokCallback,
            (XtCallbackProc) promptMessageOkCB, text);
  else if (strcmp(ext, ".nec") == 0)
    XtAddCallback (messageBox, XmNokCallback,
            (XtCallbackProc) promptMessageOkCB2, text);

  XtVaSetValues(getTopShell(messageBox), XmNmwmInputMode,
            MWM_INPUT_SYSTEM_MODAL, NULL);

  XBell (XtDisplay (messageBox), 50);
  XtManageChild (messageBox);

} /* end createPromptMessageDialog2 */

/*****************************************************************/
void newSaveOkCallback(w, text, call_data)
Widget w, text;
XmAnyCallbackStruct *call_data;
{
  Widget shell = XtParent(w);
  char title[80];
  extern char *necInputFilename;
  extern void clearDataInputs ();
  extern void momExportAction();

  if (!textValidCheck(text))
    return;
  if (necInputFilename && necInputFilename[0])
    momExportAction(necInputFilename);

  clearDataInputs();
```

```
  sprintf(title, "NEEDS %s - [%s]", VERSION, "New File");
  XtVaSetValues(topLevel, XmNtitle, title, NULL);
  XtDestroyWidget(shell);

  call_data = NULL;

} /* end newSaveOkCallback */

/*****************************************************************/
void newSaveCancelCallback(w, client_data, call_data)
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *call_data;
{
 char title[80];
 Widget shell = XtParent(w);
 extern void clearDataInputs ();

 clearDataInputs();
 sprintf(title, "NEEDS %s - [%s]", VERSION, "New File");
 XtVaSetValues(topLevel, XmNtitle, title, NULL);
 XtDestroyWidget(shell);

 client_data = NULL;
 call_data = NULL;

} /* end exitCancelCallback */

/*****************************************************************/
void createNewSaveDialog (parent, title, message)
 Widget      parent;
 char        *title,
             *message;
{
 Widget      messageBox, text;
 XmString    string,
             okString,
             cancelString,
             xmtitle;
 Arg         args [10];
 int         n = 0;
 extern char *necInputFilename;
 extern Widget getTopShell();

 string = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
 okString = XmStringCreateSimple ("OK");
 cancelString = XmStringCreateSimple ("Cancel");
 xmtitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);
 XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
 XtSetArg (args [n], XmNmessageString, string); n++;
 XtSetArg (args [n], XmNautoUnmanage, False); n++;
 XtSetArg (args [n], XmNcancelLabelString, cancelString); n++;
 XtSetArg (args [n], XmNokLabelString, okString); n++;
 XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;

 messageBox = XmCreateMessageDialog (parent, "Message", args, n);

 text = XtCreateManagedWidget("text", xmTextWidgetClass,
                messageBox, NULL, 0);
 if (necInputFilename && necInputFilename[0])
  XmTextSetString(text, necInputFilename);
 else
  XmTextSetString(text, "untitled.nec");
 XtAddCallback (messageBox, XmNokCallback,
          (XtCallbackProc) newSaveOkCallback, text);
 XtAddCallback (messageBox, XmNcancelCallback,
          (XtCallbackProc) newSaveCancelCallback, NULL);
 XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));

 XtVaSetValues (messageBox, XmNdialogType, XmDIALOG_WARNING, NULL);

 XmStringFree (string);
 XmStringFree (okString);
 XmStringFree (cancelString);
 XmStringFree (xmtitle);

 XtVaSetValues(getTopShell(messageBox), XmNmwmInputMode,
          MWM_INPUT_SYSTEM_MODAL, NULL);

 XBell (XtDisplay (messageBox), 50);
 XtManageChild (messageBox);

} /* end createNewSaveDialog */

/*****************************************************************/
void openSaveOkCallback(w, text, call_data)
Widget w, text;
XmAnyCallbackStruct *call_data;
{
 Widget shell = getTopShell(w);
 char title[80];
 extern char *necInputFilename;
 extern void momExportAction();
```

```
  if (!textValidCheck(text))
    return;
  if (necInputFilename && necInputFilename[0])
    momExportAction(necInputFilename);

  sprintf(title, "NEEDS %s - [%s]", VERSION, necInputFilename);
  XtVaSetValues(topLevel, XmNtitle, title, NULL);
  XtDestroyWidget(shell);

  call_data = NULL;

} /* end openSaveOkCallback */

void openSaveCancelCallback(w, client_data, call_data)
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *call_data;
{
  Widget shell = getTopShell(w);

  XtDestroyWidget(shell);

  client_data = NULL;
  call_data = NULL;

} /* end openSaveCancelCallback */

void createOpenSaveDialog (parent, title, message)
  Widget       parent;
  char         *title,
               *message;
{
  Widget       messageBox, text;
  XmString     string,
               okString,
               cancelString,
               xmtitle;
  Arg          args [10];
  int          n = 0;
  extern char *necInputFilename;
  extern Widget getTopShell();

  string = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
  okString = XmStringCreateSimple ("OK");
  cancelString = XmStringCreateSimple ("Cancel");
  xmtitle = XmStringCreateLtoR (title, XmSTRING_DEFAULT_CHARSET);
  XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
  XtSetArg (args [n], XmNmessageString, string); n++;
  XtSetArg (args [n], XmNautoUnmanage, False); n++;
  XtSetArg (args [n], XmNcancelLabelString, cancelString); n++;
  XtSetArg (args [n], XmNokLabelString, okString); n++;
  XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;

  messageBox = XmCreateMessageDialog (parent, "Message", args, n);

  text = XtCreateManagedWidget("text", xmTextWidgetClass,
                        messageBox, NULL, 0);
  if (necInputFilename && necInputFilename[0])
    XmTextSetString(text, necInputFilename);
  else
    XmTextSetString(text, "untitled.nec");
  XtAddCallback (messageBox, XmNokCallback,
            (XtCallbackProc) openSaveOkCallback, text);
  XtAddCallback (messageBox, XmNcancelCallback,
            (XtCallbackProc) openSaveCancelCallback, NULL);
  XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));

  XtVaSetValues (messageBox, XmNdialogType, XmDIALOG_WARNING, NULL);

  XmStringFree (string);
  XmStringFree (okString);
  XmStringFree (cancelString);
  XmStringFree (xmtitle);

  XtVaSetValues(getTopShell(messageBox), XmNmwmInputMode,
              MWM_INPUT_SYSTEM_MODAL, NULL);

  XBell (XtDisplay (messageBox), 50);
  XtManageChild (messageBox);

} /* end createOpenSaveDialog */
```

A.5 cFileMenu.c, cFileMenu.h

cFileMenu.c:

```
/*
 * The procedures in this file are the callbacks for
 * File menu items.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/IntrinsicP.h>
#include <Xm/Xm.h>
#include <Xm/FileSB.h>
#include <Xm/Text.h>
#include <Xm/MwmUtil.h>
#include "cFileMenu.h"
#include "control.h"

#define                 FloatMalloc(x)          (float *)XtMalloc(sizeof(float) * x);
#define                 IntMalloc(x)   (int *)XtMalloc(sizeof(int) * x);

extern Widget   topLevel;
extern char *necInputFilename;
char            *inputFilename = NULL;
char            *exportFilename = NULL;
Boolean saveAlert = False;

Widget   saveFileSelectionDialog;
static Widget   openFileSelectionDialog;

/* forward declarations of procedures */

extern FILE     *efopen ();
extern char     *getFilename ();
extern Widget   createFileSelectionDialog ();
extern void     createMessageDialog();

static void     readOkCB ();
static void     saveAsOkCB ();
void                    saveInputToFile ();
int                     writeInputToFile ();
int             readInputFromFile ();
Boolean searchFile();

/*****************************************************************************
 * Clears out old data and sets the inputFilename back to "Untitled".
 */
void clearDataInputs ()
{
 extern char *necInputFilename, *necOutputFilename;
 char string[160];
 extern float FrequenciesScale [];
 extern struct link *checkList, *position[], *holdList;
 extern struct countRecord record;
 struct link *node;
 extern void updatePosition();
 extern void copyCount();
 extern struct link *copyList();
 extern Widget editCtrlShell, editGeoShell;
 extern struct glink *gcheckList;
 extern struct link *checkList;
 extern struct glink *gcheckList, *gposition[], *gholdList;
 extern struct gcountRecord grecord;
 extern void gUpdatePosition();
 extern void gCopyCount();
 extern struct glink *gCopyList();

 /* Free Comments */
 if (CommentCount)
  XtFree ((char *)CM);
 CM = NULL;
 if (SWireCount) {
  XtFree ((char *)GW_ITG);
  XtFree ((char *)GW_END1);
  XtFree ((char *)GW_END2);
  XtFree ((char *)GW_NS);
  XtFree ((char *)GW_RAD);
 }
 GW_ITG = NULL;
 GW_END1 = NULL;
 GW_END2 = NULL;
 GW_NS = NULL;
 GW_RAD = NULL;
 if (TaperWireCount) {
  XtFree ((char *)GC_ITG);
  XtFree ((char *)GC_END1);
  XtFree ((char *)GC_END2);
  XtFree ((char *)GC_NS);
  XtFree ((char *)GC_IX);
```

```
    XtFree ((char *)GC_RDEL);
    XtFree ((char *)GC_RAD1);
    XtFree ((char *)GC_RAD2);
  }
  GC_ITG = NULL;
  GC_END1 = NULL;
  GC_END2 = NULL;
  GC_NS = NULL;
  GC_IX = NULL;
  GC_RDEL = NULL;
  GC_RAD1 = NULL;
  GC_RAD2 = NULL;
  if (CWireCount) {
    XtFree ((char *) CW_ITG);
    XtFree ((char *) CW_END1);
    XtFree ((char *) CW_END2);
    XtFree ((char *) CW_NS);
    XtFree ((char *) CW_ICAT);
    XtFree ((char *) CW_RAD);
    XtFree ((char *) CW_RHM);
    XtFree ((char *) CW_ZM);
  }
  CW_ITG = NULL;
  CW_END1 = NULL;
  CW_END2 = NULL;
  CW_NS = NULL;
  CW_ICAT = NULL;
  CW_RAD = NULL;
  CW_RHM = NULL;
  CW_ZM = NULL;
  if (WireArcCount) {
    XtFree ((char *) GA_ITG);
    XtFree ((char *) GA_NS);
    XtFree ((char *) GA_RADA);
    XtFree ((char *) GA_ANG1);
    XtFree ((char *) GA_ANG2);
    XtFree ((char *) GA_RAD);
  }
  GA_ITG = NULL;
  GA_NS = NULL;
  GA_RADA = NULL;
  GA_ANG1 = NULL;
  GA_ANG2 = NULL;
  GA_RAD = NULL;
  if (HelixOrSpiralCount) {
    XtFree ((char *) GH_ITG);
    XtFree ((char *) GH_NS);
    XtFree ((char *) GH_ISPX);
    XtFree ((char *) GH_TURNS);
    XtFree ((char *) GH_ZLEN);
    XtFree ((char *) GH_HR1);
    XtFree ((char *) GH_HR2);
    XtFree ((char *) GH_WR1);
    XtFree ((char *) GH_WR2);
  }
  GH_ITG = NULL;
  GH_NS = NULL;
  GH_ISPX = NULL;
  GH_TURNS = NULL;
  GH_ZLEN = NULL;
  GH_HR1 = NULL;
  GH_HR2 = NULL;
  GH_WR1 = NULL;
  GH_WR2 = NULL;
  if (SurfacePatchCount) {
    XtFree ((char *) SP_NS);
    XtFree ((char *) SP_Corner1);
    XtFree ((char *) SP_Corner3);
    XtFree ((char *) SP_Corner4);
    XtFree ((char *) SP_Corner2);
  }
  SP_NS = NULL;
  SP_Corner1 = NULL;
  SP_Corner2 = NULL;
  SP_Corner3 = NULL;
  SP_Corner4 = NULL;
  if (MultiplePatchCount) {
    XtFree ((char *) SM_Corner1);
    XtFree ((char *) SM_Corner3);
    XtFree ((char *) SM_Corner2);
    XtFree ((char *) SM_Number12);
    XtFree ((char *) SM_Number23);
  }
  SM_Corner1 = NULL;
  SM_Corner2 = NULL;
  SM_Corner3 = NULL;
  SM_Number12 = NULL;
  SM_Number23 = NULL;
  if (TransformCount) {
    XtFree ((char *) GM_ITGI);
    XtFree ((char *) GM_NRPT);
    XtFree ((char *) GM_ROX);
    XtFree ((char *) GM_ROY);
```

```
    XtFree ((char *) GM_ROZ);
    XtFree ((char *) GM_XS);
    XtFree ((char *) GM_YS);
    XtFree ((char *) GM_ZS);
  }
  GM_ITGI = NULL;
  GM_NRPT = NULL;
  GM_ROX = NULL;
  GM_ROY = NULL;
  GM_ROZ = NULL;
  GM_XS = NULL;
  GM_YS = NULL;
  GM_ZS = NULL;
  if (RotationCount) {
    XtFree ((char *) GR_ITGI);
    XtFree ((char *) GR_NR);
  }
  GR_ITGI = NULL;
  GR_NR = NULL;
  if (ReflectionCount) {
    XtFree ((char *) GX_ITGI);
    XtFree ((char *) GX_IXYZ);
  }
  GX_ITGI = NULL;
  GX_IXYZ = NULL;
  if (FrequencyCount) {
    XtFree ((char *) FR_IFRQ);
    XtFree ((char *) FR_NFRQ);
    XtFree ((char *) FR_FMHZ);
    XtFree ((char *) FR_DELFRQ);
  }
  FR_IFRQ = NULL;
  FR_NFRQ = NULL;
  FR_FMHZ = NULL;
  FR_DELFRQ = NULL;
  if (LoadsCount) {
    XtFree ((char *) LD_LDTYP);
    XtFree ((char *) LD_Tag);
    XtFree ((char *) LD_Distance);
    XtFree ((char *) LD_Distance2);
    XtFree ((char *) LD_ZLR);
    XtFree ((char *) LD_ZLI);
    XtFree ((char *) LD_ZLC);
  }
  LD_LDTYP = NULL;
  LD_Tag = NULL;
  LD_Distance = NULL;
  LD_Distance2 = NULL;
  LD_ZLR = NULL;
  LD_ZLI = NULL;
  LD_ZLC = NULL;
  if (VoltageSourcesCount) {
    XtFree ((char *) EX_Type);
    XtFree ((char *) EX_Wire);
    XtFree ((char *) EX_Format);
    XtFree ((char *) EX_Distance);
    XtFree ((char *) EX_Magnitude);
    XtFree ((char *) EX_Phase);
    XtFree ((char *) EX_Normal);
  }
  EX_Type = NULL;
  EX_Wire = NULL;
  EX_Format = NULL;
  EX_Distance = NULL;
  EX_Magnitude = NULL;
  EX_Phase = NULL;
  EX_Normal = NULL;
  if (IncidentPlaneWaveCount) {
    XtFree ((char *) EX_TYPE);
    XtFree ((char *) EX_THETA_LO);
    XtFree ((char *) EX_THETA_STEP);
    XtFree ((char *) EX_THETA_NUM);
    XtFree ((char *) EX_PHI_LO);
    XtFree ((char *) EX_PHI_STEP);
    XtFree ((char *) EX_PHI_NUM);
    XtFree ((char *) EX_POL_ANGLE);
    XtFree ((char *) EX_POL_RATIO);
    XtFree ((char *) EX_INC_MAG);
  }
  EX_TYPE = NULL;
  EX_THETA_LO = NULL;
  EX_THETA_STEP = NULL;
  EX_THETA_NUM = NULL;
  EX_PHI_LO = NULL;
  EX_PHI_STEP = NULL;
  EX_PHI_NUM = NULL;
  EX_POL_ANGLE = NULL;
  EX_POL_RATIO = NULL;
  EX_INC_MAG = NULL;
  if (TransmissionLinesCount) {
    XtFree ((char *) TL_Wire1);
    XtFree ((char *) TL_Wire2);
    XtFree ((char *) TL_Distance1);
```

```
    XtFree ((char *) TL_Distance2);
    XtFree ((char *) TL_ZC);
    XtFree ((char *) TL_TLEN);
    XtFree ((char *) TL_Y1R);
    XtFree ((char *) TL_Y1I);
    XtFree ((char *) TL_Y2R);
    XtFree ((char *) TL_Y2I);
  }
  TL_Wire1 = NULL;
  TL_Wire2 = NULL;
  TL_Distance1 = NULL;
  TL_Distance2 = NULL;
  TL_ZC = NULL;
  TL_TLEN = NULL;
  TL_Y1R = NULL;
  TL_Y1I = NULL;
  TL_Y2R = NULL;
  TL_Y2I = NULL;
  if (TwoPortNetsCount) {
    XtFree ((char *) NT_Wire1);
    XtFree ((char *) NT_Wire2);
    XtFree ((char *) NT_Distance1);
    XtFree ((char *) NT_Distance2);
    XtFree ((char *) NT_Y11R);
    XtFree ((char *) NT_Y11I);
    XtFree ((char *) NT_Y12R);
    XtFree ((char *) NT_Y12I);
    XtFree ((char *) NT_Y22R);
    XtFree ((char *) NT_Y22I);
  }
  NT_Wire1 = NULL;
  NT_Wire2 = NULL;
  NT_Distance1 = NULL;
  NT_Distance2 = NULL;
  NT_Y11R = NULL;
  NT_Y11I = NULL;
  NT_Y12R = NULL;
  NT_Y12I = NULL;
  NT_Y22R = NULL;
  NT_Y22I = NULL;
  if (InsulatedWiresCount) {
    XtFree ((char *) IS_I1);
    XtFree ((char *) IS_ITAG);
    XtFree ((char *) IS_Distance2);
    XtFree ((char *) IS_Distance1);
    XtFree ((char *) IS_EPSR);
    XtFree ((char *) IS_SIG);
    XtFree ((char *) IS_RADI);
  }
  IS_I1 = NULL;
  IS_ITAG = NULL;
  IS_Distance2 = NULL;
  IS_Distance1 = NULL;
  IS_EPSR = NULL;
  IS_SIG = NULL;
  IS_RADI = NULL;
  if (AddGroundParamCount) {
    XtFree ((char *) GD_ICLIF);
    XtFree ((char *) GD_EPSR2);
    XtFree ((char *) GD_SIG2);
    XtFree ((char *) GD_CLT);
    XtFree ((char *) GD_CHT);
    XtFree ((char *) GD_CHT);
  }
  GD_ICLIF = NULL;
  GD_EPSR2 = NULL;
  GD_SIG2 = NULL;
  GD_CLT = NULL;
  GD_CHT = NULL;
  GD_CHT = NULL;
  if (UpperMediumParamCount) {
    XtFree ((char *) UM_EPSR);
    XtFree ((char *) UM_SIG);
  }
  UM_EPSR = NULL;
  UM_SIG = NULL;
  if (MaxCouplingCount) {
    XtFree ((char *) CP_TAG1);
    XtFree ((char *) CP_Distance1);
    XtFree ((char *) CP_TAG2);
    XtFree ((char *) CP_Distance2);
  }
  CP_TAG1 = NULL;
  CP_Distance1 = NULL;
  CP_TAG2 = NULL;
  CP_Distance2 = NULL;
  if (NearElectricCount) {
    XtFree ((char *) NE_NEAR);
    XtFree ((char *) NE_NRX);
    XtFree ((char *) NE_NRY);
    XtFree ((char *) NE_NRZ);
    XtFree ((char *) NE_XNR);
    XtFree ((char *) NE_YNR);
```

```
    XtFree ((char *) NE_ZNR);
    XtFree ((char *) NE_DXNR);
    XtFree ((char *) NE_DYNR);
    XtFree ((char *) NE_DZNR);
  }
  NE_NEAR = NULL;
  NE_NRX = NULL;
  NE_NRY = NULL;
  NE_NRZ = NULL;
  NE_XNR = NULL;
  NE_YNR = NULL;
  NE_ZNR = NULL;
  NE_DXNR = NULL;
  NE_DYNR = NULL;
  NE_DZNR = NULL;
  if (NearMagneticCount) {
    XtFree ((char *) NH_NEAR);
    XtFree ((char *) NH_NRX);
    XtFree ((char *) NH_NRY);
    XtFree ((char *) NH_NRZ);
    XtFree ((char *) NH_XNR);
    XtFree ((char *) NH_YNR);
    XtFree ((char *) NH_ZNR);
    XtFree ((char *) NH_DXNR);
    XtFree ((char *) NH_DYNR);
    XtFree ((char *) NH_DZNR);
  }
  NH_NEAR = NULL;
  NH_NRX = NULL;
  NH_NRY = NULL;
  NH_NRZ = NULL;
  NH_XNR = NULL;
  NH_YNR = NULL;
  NH_ZNR = NULL;
  NH_DXNR = NULL;
  NH_DYNR = NULL;
  NH_DZNR = NULL;
  if (RadiationPatternCount) {
    XtFree ((char *) RP_I1);
    XtFree ((char *) RP_NTH);
    XtFree ((char *) RP_NPH);
    XtFree ((char *) RP_XNDA);
    XtFree ((char *) RP_THETS);
    XtFree ((char *) RP_PHIS);
    XtFree ((char *) RP_DTH);
    XtFree ((char *) RP_DPH);
    XtFree ((char *) RP_RFLD);
    XtFree ((char *) RP_GNOR);
  }
  RP_I1 = NULL;
  RP_NTH = NULL;
  RP_NPH = NULL;
  RP_XNDA = NULL;
  RP_THETS = NULL;
  RP_PHIS = NULL;
  RP_DTH = NULL;
  RP_DPH = NULL;
  RP_RFLD = NULL;
  RP_GNOR = NULL;
  if (GroundParamCount) {
    XtFree ((char *) GN_IPERF);
    XtFree ((char *) GN_NRADL);
    XtFree ((char *) GN_EPSR);
    XtFree ((char *) GN_SIG);
    XtFree ((char *) GN_F3);
    XtFree ((char *) GN_F4);
    XtFree ((char *) GN_F5);
    XtFree ((char *) GN_F6);
  }
  GN_IPERF = NULL;
  GN_NRADL = NULL;
  GN_EPSR = NULL;
  GN_SIG = NULL;
  GN_F3 = NULL;
  GN_F4 = NULL;
  GN_F5 = NULL;
  GN_F6 = NULL;
  if (PrintChargeCount) {
    XtFree ((char *) PQ_IPTFLQ);
    XtFree ((char *) PQ_IPTAQ);
    XtFree ((char *) PQ_Distance1);
    XtFree ((char *) PQ_Distance2);
  }
  PQ_IPTFLQ = NULL;
  PQ_IPTAQ = NULL;
  PQ_Distance1 = NULL;
  PQ_Distance2 = NULL;
  if (PrintLengthCount) {
    XtFree ((char *) PS_FLG);
  }
  PS_FLG = NULL;
  if (PrintCurrentCount) {
    XtFree ((char *) PT_IPTFLQ);
```

```
  XtFree ((char *) PT_IPTAQ);
  XtFree ((char *) PT_Distance1);
  XtFree ((char *) PT_Distance2);
}
PT_IPTFLQ = NULL;
PT_IPTAQ = NULL;
PT_Distance1 = NULL;
PT_Distance2 = NULL;

/* Clear out element counts for all data arrays */
NodeCount = 0;
SWireCount = 0;
TaperWireCount = 0;
CWireCount = 0;
WireArcCount = 0;
HelixOrSpiralCount = 0;
SurfacePatchCount = 0;
MultiplePatchCount = 0;
RotationCount = 0;
ReflectionCount = 0;
TransformCount = 0;
FrequencyCount = 1;  /* Always have at least one frequency! */
InsulatedWiresCount = 0;
LoadsCount = 0;
UpperMediumParamCount = 0;
VoltageSourcesCount = 0;
IncidentPlaneWaveCount = 0;
TwoPortNetsCount = 0;
TransmissionLinesCount = 0;
MaxCouplingCount = 0;
AddGroundParamCount = 0;
NearElectricCount = 0;
NearMagneticCount = 0;
RadiationPatternCount = 0;
GroundParamCount = 0;
PrintChargeCount = 0;
PrintCurrentCount = 0;
PrintLengthCount = 0;
ExecuteCount = 1;
CommentCount = 0;

/* Set environment, dimension & frequency unit to defaults */
FrequencyIndex = MHZ;
EnvIndex = FREE_SPACE;
DimIndex = METERS;

/* Set Frequency to defaults */
FR_IFRQ = IntMalloc (FrequencyCount);
FR_NFRQ = IntMalloc (FrequencyCount);
FR_FMHZ = FloatMalloc (FrequencyCount);
FR_DELFRQ = FloatMalloc (FrequencyCount);
FR_IFRQ[0] = 0;
FR_NFRQ[0] = 1;
FR_FMHZ[0] = 299.8;
FR_DELFRQ[0] = 1.0;

/* initialize link list */
checkList = emptyList(checkList);
checkList = (struct link *)XtMalloc(sizeof(struct link));
checkList->string = NULL;
checkList->prev = NULL;
checkList->next = NULL;
node = checkList;
/* FR card */
node->next = (struct link *)XtMalloc(sizeof(struct link));
node->next->prev = node;
node = node->next;
node->next = NULL;
node->tableType = FR;
node->table.fr.fr_ifrq = FR_IFRQ[0];
node->table.fr.fr_nfrq = FR_NFRQ[0];
node->table.fr.fr_fmhz = FR_FMHZ[0];
node->table.fr.fr_delfrq = FR_DELFRQ[0];
sprintf (string, "FR %d, %d, 0, 0, %.3f, %.3f", FR_IFRQ[0],
    FR_NFRQ[0], FR_FMHZ[0] * FrequenciesScale[FrequencyIndex],
    FR_DELFRQ[0]);
node->string = XmStringCreateSimple (string);
/* XQ card */
node->next = (struct link *)XtMalloc(sizeof(struct link));
node->next->prev = node;
node = node->next;
node->next = NULL;
node->tableType = XQ;
sprintf (string, "XQ");
node->string = XmStringCreateSimple (string);
/* EN card */
node->next = (struct link *)XtMalloc(sizeof(struct link));
node->next->prev = node;
node = node->next;
node->next = NULL;
node->tableType = EN;
sprintf (string, "EN");
node->string = XmStringCreateSimple (string);
```

```
    updatePosition(position, checkList);
    copyCount(&record);
    holdList = copyList(checkList, holdList);

    gcheckList = gEmptyList(gcheckList);
    gUpdatePosition(gposition, gcheckList);
    gCopyCount(&grecord);
    gholdList = gCopyList(gcheckList, gholdList);

    /* Set necInputFilename to NULL */
    if (necInputFilename != NULL) {
      XtFree (necInputFilename);
      necInputFilename = NULL;
    }
    if (necOutputFilename != NULL) {
      XtFree (necOutputFilename);
      necOutputFilename = NULL;
    }
    if (inputFilename != NULL) {
      XtFree (inputFilename);
      inputFilename = NULL;
    }
    saveAlert = False;

  } /* end clearDataInputs */

  /*****************************************************************/
  void exitNeeds ()
  {
    extern void createExitDialog();
    extern void saveInitEnv();
    extern Boolean sphigsOff;
    char *msg = "Input data have been modified.\nSave NEC data file (*.nec) ?";

    if (saveAlert)
      createExitDialog(topLevel, "Warning", msg);

    else {
      /* Save current *.mom into init file */
      saveInitEnv(necInputFilename);

      /* Disable SRGP */
      if (!sphigsOff) SRGP_end ();

      XtCloseDisplay (XtDisplay (topLevel));
      exit (0);
    }

  } /* end exitNeeds */

  /*****************************************************************/
  void openSaveAsWindow ()
  {
    Widget   w;
    XmString dirmask;
    Widget list;
    extern Widget getTopShell();

    if (saveFileSelectionDialog == NULL) {
      saveFileSelectionDialog = createFileSelectionDialog ("Save As", "*.nec");
      XtAddCallback (saveFileSelectionDialog, XmNokCallback, saveAsOkCB, NULL);
      w = XmFileSelectionBoxGetChild (saveFileSelectionDialog, XmDIALOG_TEXT);
      XmTextSetString (w, necInputFilename);
      XtVaSetValues(getTopShell(saveFileSelectionDialog),
              XmNmwmInputMode, MWM_INPUT_SYSTEM_MODAL, NULL);
    }
    XtManageChild (saveFileSelectionDialog);

    list = XmFileSelectionBoxGetChild(saveFileSelectionDialog,
                        XmDIALOG_LIST);
    XmListDeselectAllItems(list);
    dirmask = XmStringLtoRCreate("*.nec", XmSTRING_DEFAULT_CHARSET);
    XmFileSelectionDoSearch(saveFileSelectionDialog, dirmask);
    XmStringFree(dirmask);

  } /* end openSaveAsWindow */

  /*****************************************************************
   * Reads the NEEDS input data from a file.
   */
  void openReadInputWindow ()
  {
    XmString dirmask;
    Widget list;
    char *msg = "Input data have been modified.\nSave NEC file (*.nec) ?";
    extern void createOpenSaveDialog();

    if (openFileSelectionDialog == NULL) {
      openFileSelectionDialog = createFileSelectionDialog ("Open", "*.nec");

      XtAddCallback (openFileSelectionDialog, XmNokCallback, readOkCB, NULL);
    }
```

```
  XtManageChild (openFileSelectionDialog);

  list = XmFileSelectionBoxGetChild(openFileSelectionDialog,
                        XmDIALOG_LIST);
  XmListDeselectAllItems(list);
  dirmask = XmStringLtoRCreate("*.nec", XmSTRING_DEFAULT_CHARSET);
  XmFileSelectionDoSearch(openFileSelectionDialog, dirmask);
  XmStringFree(dirmask);


  if (saveAlert) {
    createOpenSaveDialog(topLevel, "Warning", msg);
    saveAlert = False;
  }

} /* end openReadInputWindow */

/*****************************************************************/
void saveInputToFile ()
{
  extern void momExportAction();

  if (necInputFilename == NULL)
    openSaveAsWindow ();
  else {
    momExportAction(necInputFilename);
  }

  saveAlert = False;

} /* end saveInputToFile */

/*****************************************************************
 * Save the inputFilename and then reads the NEEDS input data into
 * the appropriate data structures.
 */
static void readOkCB (w, clientData, callData)
  Widget w;
  XtPointer clientData;
  XmFileSelectionBoxCallbackStruct *callData;
{
  Widget dialog;
  char *filename;
  extern char *necInputFilename;
  extern char *necOutputFilename;
  char *tail, newname [80];
  char title[80], msg[80];
  extern void readNecFile();
  extern Widget  createWorkingDialog ();
  extern Widget editGeoShell, editCtrlShell;

  if ((filename = getFilename (callData)) == NULL)
    return;

  if (!strstr(filename, ".nec")) {
    sprintf(msg, "Improper NEC file extension [%s]", filename);
    createMessageDialog(topLevel, "Warning", msg, XmDIALOG_WARNING);
    return;
  }
  closeAll();
  dialog = createWorkingDialog (w, "Message",
        "Importing files ...");
  XtManageChild (dialog);
  forceUpdate (dialog);
  readNecFile(filename);
  XtDestroyWidget (dialog);
  XtFree (necInputFilename);
  necInputFilename = filename;
  sprintf(title, "NEEDS %s - [%s]", VERSION, filename);
  XtVaSetValues(topLevel, XmNtitle, title, NULL);

  XtUnmanageChild (openFileSelectionDialog);

  strcpy (newname, filename);
  tail = strstr (newname, ".nec");
  strcpy (tail, ".out");
  XtFree((char *)necOutputFilename);
  necOutputFilename = XtMalloc(strlen(newname) + 1);
  strcpy(necOutputFilename, newname);

  clientData = NULL;

} /* end readOkCB */

/*****************************************************************
 * Saves the inputFilename and then saves the data to a file called
 * "inputFilename".
 */
static void saveAsOkCB (w, clientData, callData)
  Widget w;
  XtPointer clientData;
  XmFileSelectionBoxCallbackStruct *callData;
{
```

```
  char *filename;
  extern Widget inputFilenameText;
  char title[80], msg[80];
  char *tail;
  XEvent event;
  Window win = XtWindow(topLevel);
  XtAppContext cxt = XtWidgetToApplicationContext(w);
  extern char *exportFilename, *necInputFilename, *necOutputFilename;
  extern void momExportAction();

  if ((filename = getFilename (callData)) == NULL)
    return;
  if (!strstr(filename, ".nec")) {
    XtUnmanageChild (saveFileSelectionDialog);
    sprintf(msg, "Incorrect NEC file extension [%s] ?", filename);
    createMessageDialog(topLevel, "Warning", msg, XmDIALOG_WARNING);
    while (True) {
      XtAppNextEvent(cxt, &event);
      XtDispatchEvent(&event);
      if (event.xfocus.type == FocusIn && event.xfocus.window == win) {
        XtManageChild (saveFileSelectionDialog);
        break;
      }
    }
    return;
  }
  momExportAction(filename);
  XtFree (necInputFilename);
  necInputFilename = filename;
  sprintf(title, "NEEDS %s - [%s]", VERSION, filename);
  XtVaSetValues(topLevel, XmNtitle, title, NULL);
  XtUnmanageChild (saveFileSelectionDialog);
  saveAlert = False;

  strcpy(title, filename);
  tail = strstr(title, ".nec");
  strcpy(tail, ".out");
  XtFree(necOutputFilename);
  necOutputFilename = XtMalloc(strlen(title) + 1);
  strcpy(necOutputFilename, title);

  w = w;  /* Make compiler happy */
  clientData = clientData;

} /* end saveAsOkCB */

/*****************************************************************
 * Opens "inputFilename" and reads NEEDS input data from it
 */
int readInputFromFile (filename)
  char  *filename;
{
  FILE *fp;
  int i;
  char string [132];
  char name [132];
  void clearOldData();
  extern void createMessageDialog2();

  if ((fp = efopen (filename, "r")) == NULL)
    return (0);                 /* return 0 for failed write */

  if (fgetc(fp) == EOF) { /* empty file */
    char msg[80];

    /* clear all the previous data read from the previous file */
    clearOldData();
    sprintf(msg, "Data file %s is empty", filename);
    createMessageDialog2(topLevel, "Warning", msg, XmDIALOG_WARNING, NULL);
    fclose(fp);
    remove(filename);
    return (1);
  }
  else
    rewind(fp);

  /* Read environment, dimension & frequency unit data */
  fscanf (fp, "Environment %d\n", &EnvIndex);
  fscanf (fp, "Dimension %d\n", &DimIndex);
  fscanf (fp, "Freq Unit %d\n", &FrequencyIndex);

  /* Read Comments data */
  fscanf (fp, "Comments %d\n", &CommentCount);
  if (CommentCount) {
    char *token, string[160];
    XtFree((char *)CM);
    CM = (stringType *)XtMalloc(sizeof(stringType) * CommentCount);
    for (i = 0; i < CommentCount; i++) {
      fgets (string, 160, fp);
      token = strtok(string, "\n");
      if (token)
        strcpy(CM[i].line, token);
    }
```

```
} else {
  XtFree((char *)CM);
  CM = NULL;
}

/* Read Node Coordinates data */
fscanf (fp, "\nNodes %d\n", &NodeCount);

/* Allocate X, Y, and Z arrays */
XtFree ((char *) X);
X = NULL;
XtFree ((char *) Y);
Y = NULL;
XtFree ((char *) Z);
Z = NULL;
if (NodeCount) {
  X = FloatMalloc (NodeCount);
  Y = FloatMalloc (NodeCount);
  Z = FloatMalloc (NodeCount);
}

for (i = 0; i < NodeCount; i++)
  fscanf (fp, "%f %f %f\n", &X[i], &Y[i], &Z[i]);

/* Read in Straight Wires data */
fscanf (fp, "Straight Wires %d\n", &SWireCount);
XtFree ((char *) GW_ITG);
GW_ITG = NULL;
XtFree ((char *) GW_END1);
GW_END1 = NULL;
XtFree ((char *) GW_END2);
GW_END2 = NULL;
XtFree ((char *) GW_NS);
GW_NS = NULL;
XtFree ((char *) GW_RAD);
GW_RAD = NULL;
if (SWireCount) {
  GW_ITG = IntMalloc (SWireCount);
  GW_END1 = IntMalloc (SWireCount);
  GW_END2 = IntMalloc (SWireCount);
  GW_NS = IntMalloc (SWireCount);
  GW_RAD = FloatMalloc (SWireCount);
}
for (i = 0; i < SWireCount; i++)
  fscanf (fp, "%d %d %d %d %f\n", &GW_ITG[i], &GW_END1[i], &GW_END2[i],
     &GW_NS[i], &GW_RAD[i]);

/* Read in Tapered Wires data */
fscanf (fp, "Tapered Wires %d\n", &TaperWireCount);
XtFree ((char *) GC_ITG);
GC_ITG = NULL;
XtFree ((char *) GC_END1);
GC_END1 = NULL;
XtFree ((char *) GC_END2);
GC_END2 = NULL;
XtFree ((char *) GC_NS);
GC_NS = NULL;
XtFree ((char *) GC_IX);
GC_IX = NULL;
XtFree ((char *) GC_RDEL);
GC_RDEL = NULL;
XtFree ((char *) GC_RAD1);
GC_RAD1 = NULL;
XtFree ((char *) GC_RAD2);
GC_RAD2 = NULL;
XtFree ((char *) GC_DEL1);
GC_DEL1 = NULL;
XtFree ((char *) GC_DEL2);
GC_DEL2 = NULL;
if (TaperWireCount) {
  GC_ITG = IntMalloc (TaperWireCount);
  GC_END1 = IntMalloc (TaperWireCount);
  GC_END2 = IntMalloc (TaperWireCount);
  GC_NS = IntMalloc (TaperWireCount);
  GC_IX = IntMalloc (TaperWireCount);
  GC_RDEL = FloatMalloc (TaperWireCount);
  GC_RAD1 = FloatMalloc (TaperWireCount);
  GC_RAD2 = FloatMalloc (TaperWireCount);
  GC_DEL1 = FloatMalloc (TaperWireCount);
  GC_DEL2 = FloatMalloc (TaperWireCount);
}
for (i = 0; i < TaperWireCount; i++)
  fscanf (fp, "%d %d %d %d %d %f %f %f %f\n", &GC_ITG[i], &GC_END1[i],
     &GC_END2[i], &GC_NS[i], &GC_IX[i], &GC_RDEL[i], &GC_RAD1[i],
     &GC_RAD2[i], &GC_DEL1[i], &GC_DEL2[i]);

/* Read in Cantenary Wires data */
fscanf (fp, "Cantenary Wires %d\n", &CWireCount);
XtFree ((char *) CW_ITG);
CW_ITG = NULL;
XtFree ((char *) CW_END1);
CW_END1 = NULL;
XtFree ((char *) CW_END2);
```

```
  CW_END2 = NULL;
  XtFree ((char *) CW_NS);
  CW_NS = NULL;
  XtFree ((char *) CW_RAD);
  CW_RAD = NULL;
  XtFree ((char *) CW_ICAT);
  CW_ICAT = NULL;
  XtFree ((char *) CW_RHM);
  CW_RHM = NULL;
  XtFree ((char *) CW_ZM);
  CW_ZM = NULL;
  if (CWireCount) {
   CW_ITG = IntMalloc (CWireCount);
   CW_END1 = IntMalloc (CWireCount);
   CW_END2 = IntMalloc (CWireCount);
   CW_NS = IntMalloc (CWireCount);
   CW_RAD = FloatMalloc (CWireCount);
   CW_ICAT = IntMalloc (CWireCount);
   CW_RHM = FloatMalloc (CWireCount);
   CW_ZM = FloatMalloc (CWireCount);
  }
  for (i = 0; i < CWireCount; i++)
   fscanf (fp, "%d %d %d %d %f %d %f %f\n", &CW_ITG[i], &CW_END1[i],
     &CW_END2[i], &CW_NS[i], &CW_RAD[i], &CW_ICAT[i], &CW_RHM[i],
     &CW_ZM[i]);

  /* Read in Wire Arc data */
  fscanf (fp, "Wire Arc %d\n", &WireArcCount);
  XtFree ((char *) GA_ITG);
  GA_ITG = NULL;
  XtFree ((char *) GA_NS);
  GA_NS = NULL;
  XtFree ((char *) GA_RADA);
  GA_RADA = NULL;
  XtFree ((char *) GA_ANG1);
  GA_ANG1 = NULL;
  XtFree ((char *) GA_ANG2);
  GA_ANG2 = NULL;
  XtFree ((char *) GA_RAD);
  GA_RAD = NULL;
  if (WireArcCount) {
   GA_ITG = IntMalloc (WireArcCount);
   GA_NS = IntMalloc (WireArcCount);
   GA_RADA = FloatMalloc (WireArcCount);
   GA_ANG1 = FloatMalloc (WireArcCount);
   GA_ANG2 = FloatMalloc (WireArcCount);
   GA_RAD = FloatMalloc (WireArcCount);
  }
  for (i = 0; i < WireArcCount; i++)
   fscanf (fp, "%d %d %f %f %f %f\n", &GA_ITG[i], &GA_NS[i], &GA_RADA[i],
     &GA_ANG1[i], &GA_ANG2[i], &GA_RAD[i]);

  /* Read in Helix & Spiral data */
  fscanf (fp, "Helix/Spiral %d\n", &HelixOrSpiralCount);
  XtFree ((char *) GH_ITG);
  GH_ITG = NULL;
  XtFree ((char *) GH_NS);
  GH_NS = NULL;
  XtFree ((char *) GH_TURNS);
  GH_TURNS = NULL;
  XtFree ((char *) GH_ZLEN);
  GH_ZLEN = NULL;
  XtFree ((char *) GH_HR1);
  GH_HR1 = NULL;
  XtFree ((char *) GH_HR2);
  GH_HR2 = NULL;
  XtFree ((char *) GH_WR1);
  GH_WR1 = NULL;
  XtFree ((char *) GH_WR2);
  GH_WR2 = NULL;
  XtFree ((char *) GH_ISPX);
  GH_ISPX = NULL;
  if (HelixOrSpiralCount) {
   GH_ITG = IntMalloc (HelixOrSpiralCount);
   GH_NS = IntMalloc (HelixOrSpiralCount);
   GH_TURNS = FloatMalloc (HelixOrSpiralCount);
   GH_ZLEN = FloatMalloc (HelixOrSpiralCount);
   GH_HR1 = FloatMalloc (HelixOrSpiralCount);
   GH_HR2 = FloatMalloc (HelixOrSpiralCount);
   GH_WR1 = FloatMalloc (HelixOrSpiralCount);
   GH_WR2 = FloatMalloc (HelixOrSpiralCount);
   GH_ISPX = FloatMalloc (HelixOrSpiralCount);
  }
  for (i = 0; i < HelixOrSpiralCount; i++)
   fscanf (fp, "%d %d %f %f %f %f %f %f\n", &GH_ITG[i],
     &GH_NS[i], &GH_TURNS[i], &GH_ZLEN[i], &GH_HR1[i], &GH_HR2[i],
     &GH_WR1[i], &GH_WR2[i], &GH_ISPX[i]);

  /* Read in Surface Patch data */
  fscanf (fp, "Surface Patch %d\n", &SurfacePatchCount);
  XtFree ((char *) SP_NS);
  SP_NS = NULL;
  XtFree ((char *) SP_Corner1);
```

```
 SP_Corner1 = NULL;
 XtFree ((char *) SP_Corner2);
 SP_Corner2 = NULL;
 XtFree ((char *) SP_Corner3);
 SP_Corner3 = NULL;
 XtFree ((char *) SP_Corner4);
 SP_Corner4 = NULL;
 if (SurfacePatchCount) {
  SP_NS = IntMalloc (SurfacePatchCount);
  SP_Corner1 = IntMalloc (SurfacePatchCount);
  SP_Corner2 = IntMalloc (SurfacePatchCount);
  SP_Corner3 = IntMalloc (SurfacePatchCount);
  SP_Corner4 = IntMalloc (SurfacePatchCount);
 }
 for (i = 0; i < SurfacePatchCount; i++)
  fscanf (fp, "%d %d %d %d %d\n", &SP_NS[i],
    &SP_Corner1[i], &SP_Corner2[i], &SP_Corner3[i], &SP_Corner4[i]);

 /* Read in Multiple Patch data */
 fscanf (fp, "Multiple Patch %d\n", &MultiplePatchCount);
 XtFree ((char *) SM_Corner1);
 SM_Corner1 = NULL;
 XtFree ((char *) SM_Corner2);
 SM_Corner2 = NULL;
 XtFree ((char *) SM_Corner3);
 SM_Corner3 = NULL;
 XtFree ((char *) SM_Number12);
 SM_Number12 = NULL;
 XtFree ((char *) SM_Number23);
 SM_Number23 = NULL;
 if (MultiplePatchCount) {
  SM_Corner1 = IntMalloc (MultiplePatchCount);
  SM_Corner2 = IntMalloc (MultiplePatchCount);
  SM_Corner3 = IntMalloc (MultiplePatchCount);
  SM_Number12 = IntMalloc (MultiplePatchCount);
  SM_Number23 = IntMalloc (MultiplePatchCount);
 }
 for (i = 0; i < MultiplePatchCount; i++)
  fscanf (fp, "%d %d %d %d %d\n",
    &SM_Corner1[i], &SM_Corner2[i], &SM_Corner3[i],
    &SM_Number12[i], &SM_Number23[i]);

 /* Read in Transformations data */
 fscanf (fp, "Transform %d\n", &TransformCount);
 XtFree ((char *) GM_ITGI);
 GM_ITGI = NULL;
 XtFree ((char *) GM_NRPT);
 GM_NRPT = NULL;
 XtFree ((char *) GM_ROX);
 GM_ROX = NULL;
 XtFree ((char *) GM_ROY);
 GM_ROY = NULL;
 XtFree ((char *) GM_ROZ);
 GM_ROZ = NULL;
 XtFree ((char *) GM_XS);
 GM_XS = NULL;
 XtFree ((char *) GM_YS);
 GM_YS = NULL;
 XtFree ((char *) GM_ZS);
 GM_ZS = NULL;
 if (TransformCount) {
  GM_ITGI = IntMalloc (TransformCount);
  GM_NRPT = IntMalloc (TransformCount);
  GM_ROX = FloatMalloc (TransformCount);
  GM_ROY = FloatMalloc (TransformCount);
  GM_ROZ = FloatMalloc (TransformCount);
  GM_XS = FloatMalloc (TransformCount);
  GM_YS = FloatMalloc (TransformCount);
  GM_ZS = FloatMalloc (TransformCount);
 }
 for (i = 0; i < TransformCount; i++)
  fscanf (fp, "%d %d %f %f %f %f %f %f\n",
    &GM_ITGI[i], &GM_NRPT[i], &GM_ROX[i], &GM_ROY[i], &GM_ROZ[i],
    &GM_XS[i], &GM_YS[i], &GM_ZS[i]);

 /* Read in Rotation data */
 fscanf (fp, "Rotation %d\n", &RotationCount);
 XtFree ((char *) GR_ITGI);
 GR_ITGI = NULL;
 XtFree ((char *) GR_NR);
 GR_NR = NULL;
 if (RotationCount) {
  GR_ITGI = IntMalloc (RotationCount);
  GR_NR = IntMalloc (RotationCount);
 }
 for (i = 0; i < RotationCount; i++)
  fscanf (fp, "%d %d\n", &GR_ITGI[i], &GR_NR[i]);

 /* Read in Reflection data */
 fscanf (fp, "Reflection %d\n", &ReflectionCount);
 XtFree ((char *) GX_ITGI);
 GX_ITGI = NULL;
 XtFree ((char *) GX_IXYZ);
```

```
  GX_IXYZ = NULL;
  if (ReflectionCount) {
    GX_ITGI = IntMalloc (ReflectionCount);
    GX_IXYZ = IntMalloc (ReflectionCount);
  }
  for (i = 0; i < ReflectionCount; i++)
    fscanf (fp, "%d %d\n", &GX_ITGI[i], &GX_IXYZ[i]);

  /* Read in Frequency data */
  fscanf (fp, "Frequency %d\n", &FrequencyCount);
  XtFree ((char *) FR_IFRQ);
  FR_IFRQ = NULL;
  XtFree ((char *) FR_NFRQ);
  FR_NFRQ = NULL;
  XtFree ((char *) FR_FMHZ);
  FR_FMHZ = NULL;
  XtFree ((char *) FR_DELFRQ);
  FR_DELFRQ = NULL;
  if (FrequencyCount > 0) {
    FR_IFRQ = IntMalloc (FrequencyCount);
    FR_NFRQ = IntMalloc (FrequencyCount);
    FR_FMHZ = FloatMalloc (FrequencyCount);
    FR_DELFRQ = FloatMalloc (FrequencyCount);
    for (i = 0; i < FrequencyCount; i++)
      fscanf (fp, "%d %d %f %f\n", &FR_IFRQ[i], &FR_NFRQ[i],
          &FR_FMHZ[i], &FR_DELFRQ[i]);
  } else {
    FrequencyCount = 1;
    FR_IFRQ = IntMalloc (FrequencyCount);
    FR_NFRQ = IntMalloc (FrequencyCount);
    FR_FMHZ = FloatMalloc (FrequencyCount);
    FR_DELFRQ = FloatMalloc (FrequencyCount);
    FR_IFRQ[0] = 0;
    FR_NFRQ[0] = 1;
    FR_FMHZ[0] = 299.8;
    FR_DELFRQ[0] = 0;
  }

  /* Read in Impedance Loading */
  fscanf (fp, "Load %d\n", &LoadsCount);
  XtFree ((char *) LD_LDTYP);
  LD_LDTYP = NULL;
  XtFree ((char *) LD_Tag);
  LD_Tag = NULL;
  XtFree ((char *) LD_Distance);
  LD_Distance = NULL;
  XtFree ((char *) LD_Distance2);
  LD_Distance2 = NULL;
  XtFree ((char *) LD_ZLR);
  LD_ZLR = NULL;
  XtFree ((char *) LD_ZLI);
  LD_ZLI = NULL;
  XtFree ((char *) LD_ZLC);
  LD_ZLC = NULL;
  if (LoadsCount > 0) {
    LD_LDTYP = IntMalloc (LoadsCount);
    LD_Tag = IntMalloc (LoadsCount);
    LD_Distance = FloatMalloc (LoadsCount);
    LD_Distance2 = FloatMalloc (LoadsCount);
    LD_ZLR = FloatMalloc (LoadsCount);
    LD_ZLI = FloatMalloc (LoadsCount);
    LD_ZLC = FloatMalloc (LoadsCount);
  }
  for (i = 0; i < LoadsCount; i++)
    fscanf (fp, "%d %d %f %f %f %f %f\n", &LD_LDTYP[i], &LD_Tag[i],
      &LD_Distance[i], &LD_Distance2[i], &LD_ZLR[i], &LD_ZLI[i],
      &LD_ZLC[i]);

  /* Read in Voltage Source data */
  fscanf (fp, "Voltage Sources %d\n", &VoltageSourcesCount);
  XtFree ((char *) EX_Type);
  EX_Type = NULL;
  XtFree ((char *) EX_Wire);
  EX_Wire = NULL;
  XtFree ((char *) EX_Format);
  EX_Format = NULL;
  XtFree ((char *) EX_Distance);
  EX_Distance = NULL;
  XtFree ((char *) EX_Magnitude);
  EX_Magnitude = NULL;
  XtFree ((char *) EX_Phase);
  EX_Phase = NULL;
  XtFree ((char *) EX_Normal);
  EX_Normal = NULL;
  if (VoltageSourcesCount > 0) {
    EX_Type = IntMalloc (VoltageSourcesCount);
    EX_Wire = IntMalloc (VoltageSourcesCount);
    EX_Format = IntMalloc (VoltageSourcesCount);
    EX_Distance = FloatMalloc (VoltageSourcesCount);
    EX_Magnitude = FloatMalloc (VoltageSourcesCount);
    EX_Phase = FloatMalloc (VoltageSourcesCount);
    EX_Normal = FloatMalloc (VoltageSourcesCount);
  }
```

```
for (i = 0; i < VoltageSourcesCount; i++)
  fscanf (fp, "%d %d %d %f %f %f %f\n", &EX_Type[i], &EX_Wire[i],
       &EX_Format[i], &EX_Distance[i], &EX_Magnitude[i], &EX_Phase[i],
       &EX_Normal[i]);

/* Read in Incident Plane Wave data */
fscanf (fp, "Incident Plane Wave %d\n", &IncidentPlaneWaveCount);
XtFree ((char *) EX_TYPE);
EX_TYPE = NULL;
XtFree ((char *) EX_THETA_NUM);
EX_THETA_NUM = NULL;
XtFree ((char *) EX_PHI_NUM);
EX_PHI_NUM = NULL;
XtFree ((char *) EX_THETA_LO);
EX_THETA_LO = NULL;
XtFree ((char *) EX_PHI_LO);
EX_PHI_LO = NULL;
XtFree ((char *) EX_THETA_STEP);
EX_THETA_STEP = NULL;
XtFree ((char *) EX_PHI_STEP);
EX_PHI_STEP = NULL;
XtFree ((char *) EX_POL_ANGLE);
EX_POL_ANGLE = NULL;
XtFree ((char *) EX_POL_RATIO);
EX_POL_RATIO = NULL;
XtFree ((char *) EX_INC_MAG);
EX_INC_MAG = NULL;
if (IncidentPlaneWaveCount > 0) {
  EX_TYPE = IntMalloc (IncidentPlaneWaveCount);
  EX_THETA_NUM = IntMalloc (IncidentPlaneWaveCount);
  EX_PHI_NUM = IntMalloc (IncidentPlaneWaveCount);
  EX_THETA_LO = FloatMalloc (IncidentPlaneWaveCount);
  EX_PHI_LO = FloatMalloc (IncidentPlaneWaveCount);
  EX_THETA_STEP = FloatMalloc (IncidentPlaneWaveCount);
  EX_PHI_STEP = FloatMalloc (IncidentPlaneWaveCount);
  EX_POL_ANGLE = FloatMalloc (IncidentPlaneWaveCount);
  EX_POL_RATIO = FloatMalloc (IncidentPlaneWaveCount);
  EX_INC_MAG = FloatMalloc (IncidentPlaneWaveCount);
}
for (i = 0; i < IncidentPlaneWaveCount; i++)
  fscanf (fp, "%d %d %d %f %f %f %f %f %f %f\n", &EX_TYPE[i],
       &EX_THETA_NUM[i], &EX_PHI_NUM[i], &EX_THETA_LO[i], &EX_PHI_LO[i],
       &EX_THETA_STEP[i], &EX_PHI_STEP[i], &EX_POL_ANGLE[i],
       &EX_POL_RATIO[i], &EX_INC_MAG[i]);

/* Read in Transmission Line data */
fscanf (fp, "Transmission Lines %d\n", &TransmissionLinesCount);
XtFree ((char *) TL_Wire1);
TL_Wire1 = NULL;
XtFree ((char *) TL_Wire2);
TL_Wire2 = NULL;
XtFree ((char *) TL_Distance1);
TL_Distance1 = NULL;
XtFree ((char *) TL_Distance2);
TL_Distance2 = NULL;
XtFree ((char *) TL_ZC);
TL_ZC = NULL;
XtFree ((char *) TL_TLEN);
TL_TLEN = NULL;
XtFree ((char *) TL_Y1R);
TL_Y1R = NULL;
XtFree ((char *) TL_Y1I);
TL_Y1I = NULL;
XtFree ((char *) TL_Y2R);
TL_Y2R = NULL;
XtFree ((char *) TL_Y2I);
TL_Y2I = NULL;
if (TransmissionLinesCount > 0) {
  TL_Wire1 = IntMalloc (TransmissionLinesCount);
  TL_Wire2 = IntMalloc (TransmissionLinesCount);
  TL_Distance1 = FloatMalloc (TransmissionLinesCount);
  TL_Distance2 = FloatMalloc (TransmissionLinesCount);
  TL_ZC = FloatMalloc (TransmissionLinesCount);
  TL_TLEN = FloatMalloc (TransmissionLinesCount);
  TL_Y1R = FloatMalloc (TransmissionLinesCount);
  TL_Y1I = FloatMalloc (TransmissionLinesCount);
  TL_Y2R = FloatMalloc (TransmissionLinesCount);
  TL_Y2I = FloatMalloc (TransmissionLinesCount);
}
for (i = 0; i < TransmissionLinesCount; i++)
  fscanf (fp, "%d %d %f %f %f %f %f %f %f %f\n", &TL_Wire1[i], &TL_Wire2[i],
       &TL_Distance1[i], &TL_Distance2[i], &TL_ZC[i], &TL_TLEN[i],
       &TL_Y1R[i], &TL_Y1I[i], &TL_Y2R[i], &TL_Y2I[i]);

/* Read in Two Port Networks data */
fscanf (fp, "Two Port Nets %d\n", &TwoPortNetsCount);
XtFree ((char *) NT_Wire1);
NT_Wire1 = NULL;
XtFree ((char *) NT_Wire2);
NT_Wire2 = NULL;
XtFree ((char *) NT_Distance1);
NT_Distance1 = NULL;
XtFree ((char *) NT_Distance2);
```

```
 NT_Distance2 = NULL;
 XtFree ((char *) NT_Y11R);
 NT_Y11R = NULL;
 XtFree ((char *) NT_Y11I);
 NT_Y11I = NULL;
 XtFree ((char *) NT_Y12R);
 NT_Y12R = NULL;
 XtFree ((char *) NT_Y12I);
 NT_Y12I = NULL;
 XtFree ((char *) NT_Y22R);
 NT_Y22R = NULL;
 XtFree ((char *) NT_Y22I);
 NT_Y22I = NULL;
 if (TwoPortNetsCount > 0) {
  NT_Wire1 = IntMalloc (TwoPortNetsCount);
  NT_Wire2 = IntMalloc (TwoPortNetsCount);
  NT_Distance1 = FloatMalloc (TwoPortNetsCount);
  NT_Distance2 = FloatMalloc (TwoPortNetsCount);
  NT_Y11R = FloatMalloc (TwoPortNetsCount);
  NT_Y11I = FloatMalloc (TwoPortNetsCount);
  NT_Y12R = FloatMalloc (TwoPortNetsCount);
  NT_Y12I = FloatMalloc (TwoPortNetsCount);
  NT_Y22R = FloatMalloc (TwoPortNetsCount);
  NT_Y22I = FloatMalloc (TwoPortNetsCount);
 }
 for (i = 0; i < TwoPortNetsCount; i++)
  fscanf (fp, "%d %d %f %f %f %f %f %f %f %f\n", &NT_Wire1[i], &NT_Wire2[i],
       &NT_Distance1[i], &NT_Distance2[i], &NT_Y11R[i], &NT_Y11I[i],
       &NT_Y12R[i], &NT_Y12I[i], &NT_Y22R[i], &NT_Y22I[i]);

 /* Read in Insulated Wire data */
 fscanf (fp, "Insulated Wires %d\n", &InsulatedWiresCount);
 XtFree ((char *) IS_I1);
 IS_I1 = NULL;
 XtFree ((char *) IS_ITAG);
 IS_ITAG = NULL;
 XtFree ((char *) IS_Distance1);
 IS_Distance1 = NULL;
 XtFree ((char *) IS_Distance2);
 IS_Distance2 = NULL;
 XtFree ((char *) IS_EPSR);
 IS_EPSR = NULL;
 XtFree ((char *) IS_SIG);
 IS_SIG = NULL;
 XtFree ((char *) IS_RADI);
 IS_RADI = NULL;
 if (InsulatedWiresCount > 0) {
  IS_I1 = IntMalloc (InsulatedWiresCount);
  IS_ITAG = IntMalloc (InsulatedWiresCount);
  IS_Distance1 = FloatMalloc (InsulatedWiresCount);
  IS_Distance2 = FloatMalloc (InsulatedWiresCount);
  IS_EPSR = FloatMalloc (InsulatedWiresCount);
  IS_SIG = FloatMalloc (InsulatedWiresCount);
  IS_RADI = FloatMalloc (InsulatedWiresCount);
 }
 for (i = 0; i < InsulatedWiresCount; i++)
  fscanf (fp, "%d %d %f %f %f %f %f\n", &IS_I1[i], &IS_ITAG[i],
     &IS_Distance1[i], &IS_Distance2[i], &IS_EPSR[i], &IS_SIG[i],
     &IS_RADI[i]);

 fscanf (fp, "Ground Parameters %d\n", &GroundParamCount);
 XtFree ((char *) GN_IPERF);
 GN_IPERF = NULL;
 XtFree ((char *) GN_NRADL);
 GN_NRADL = NULL;
 XtFree ((char *) GN_EPSR);
 GN_EPSR = NULL;
 XtFree ((char *) GN_SIG);
 GN_SIG = NULL;
 XtFree ((char *) GN_F3);
 GN_F3 = NULL;
 XtFree ((char *) GN_F4);
 GN_F4 = NULL;
 XtFree ((char *) GN_F5);
 GN_F5 = NULL;
 XtFree ((char *) GN_F6);
 GN_F6 = NULL;
 if (GroundParamCount > 0) {
  GN_IPERF = IntMalloc (GroundParamCount);
  GN_NRADL = IntMalloc (GroundParamCount);
  GN_EPSR = FloatMalloc (GroundParamCount);
  GN_SIG = FloatMalloc (GroundParamCount);
  GN_F3 = FloatMalloc (GroundParamCount);
  GN_F4 = FloatMalloc (GroundParamCount);
  GN_F5 = FloatMalloc (GroundParamCount);
  GN_F6 = FloatMalloc (GroundParamCount);
 }
 for (i = 0; i < GroundParamCount; i++) {
  int j = 0;

  fgets (string, 131, fp);
  j = sscanf (string, "%d %d %f %f %f %f %f %s\n",
    &GN_IPERF[i], &GN_NRADL[i], &GN_EPSR[i], &GN_SIG[i],
```

```
    &GN_F3[i], &GN_F4[i], &GN_F5[i], &GN_F6[i], name);
   if (j == 9) {
    GN_Filename = (char *) XtMalloc (strlen (name) + 1);
    strcpy (GN_Filename, name);
   } else
    GN_Filename = NULL;
  }

  /* Read in Additional Ground Parameters data */
  fscanf (fp, "Addition Ground Parameters %d\n", &AddGroundParamCount);
  XtFree ((char *) GD_ICLIF);
  GD_ICLIF = NULL;
  XtFree ((char *) GD_EPSR2);
  GD_EPSR2 = NULL;
  XtFree ((char *) GD_SIG2);
  GD_SIG2 = NULL;
  XtFree ((char *) GD_CLT);
  GD_CLT = NULL;
  XtFree ((char *) GD_CHT);
  GD_CHT = NULL;
  if (AddGroundParamCount > 0) {
   GD_ICLIF = IntMalloc (AddGroundParamCount);
   GD_EPSR2 = FloatMalloc (AddGroundParamCount);
   GD_SIG2 = FloatMalloc (AddGroundParamCount);
   GD_CLT = FloatMalloc (AddGroundParamCount);
   GD_CHT = FloatMalloc (AddGroundParamCount);
  }
  for (i = 0; i < AddGroundParamCount; i++)
   fscanf (fp, "%d %f %f %f %f\n", &GD_ICLIF[i], &GD_EPSR2[i], &GD_SIG2[i],
        &GD_CLT[i], &GD_CHT[i]);

  /* Read in Udpper-Medium Parameters data */
  fscanf (fp, "Upper-Medium Parameters %d\n", &UpperMediumParamCount);
  XtFree ((char *) UM_EPSR);
  UM_EPSR = NULL;
  XtFree ((char *) UM_SIG);
  UM_SIG = NULL;
  if (UpperMediumParamCount > 0) {
   UM_EPSR = FloatMalloc (UpperMediumParamCount);
   UM_SIG = FloatMalloc (UpperMediumParamCount);
  }
  for (i = 0; i < UpperMediumParamCount; i++)
   fscanf (fp, "%f %f\n", &UM_EPSR[i], &UM_SIG[i]);

  /* Read in Maximum Coupling Calculation data */
  fscanf (fp, "Max Coupling %d\n", &MaxCouplingCount);
  XtFree ((char *) CP_TAG1);
  CP_TAG1 = NULL;
  XtFree ((char *) CP_TAG2);
  CP_TAG2 = NULL;
  XtFree ((char *) CP_Distance1);
  CP_Distance1 = NULL;
  XtFree ((char *) CP_Distance2);
  CP_Distance2 = NULL;
  if (MaxCouplingCount > 0) {
   CP_TAG1 = IntMalloc (MaxCouplingCount);
   CP_TAG2 = IntMalloc (MaxCouplingCount);
   CP_Distance1 = FloatMalloc (MaxCouplingCount);
   CP_Distance2 = FloatMalloc (MaxCouplingCount);
  }
  for (i = 0; i < MaxCouplingCount; i++)
   fscanf (fp, "%d %d %f %f\n", &CP_TAG1[i], &CP_TAG2[i], &CP_Distance1[i],
        &CP_Distance2[i]);

  /* Read in Near Electric Field data */
  fscanf (fp, "Near Electric Field %d\n", &NearElectricCount);
  XtFree ((char *) NE_NEAR);
  NE_NEAR = NULL;
  XtFree ((char *) NE_NRX);
  NE_NRX = NULL;
  XtFree ((char *) NE_NRY);
  NE_NRY = NULL;
  XtFree ((char *) NE_NRZ);
  NE_NRZ = NULL;
  XtFree ((char *) NE_XNR);
  NE_XNR = NULL;
  XtFree ((char *) NE_YNR);
  NE_YNR = NULL;
  XtFree ((char *) NE_ZNR);
  NE_ZNR = NULL;
  XtFree ((char *) NE_DXNR);
  NE_DXNR = NULL;
  XtFree ((char *) NE_DYNR);
  NE_DYNR = NULL;
  XtFree ((char *) NE_DZNR);
  NE_DZNR = NULL;
  if (NearElectricCount > 0) {
   NE_NEAR = IntMalloc (NearElectricCount);
   NE_NRX = IntMalloc (NearElectricCount);
   NE_NRY = IntMalloc (NearElectricCount);
   NE_NRZ = IntMalloc (NearElectricCount);
   NE_XNR = FloatMalloc (NearElectricCount);
   NE_YNR = FloatMalloc (NearElectricCount);
```

```
  NE_ZNR = FloatMalloc (NearElectricCount);
  NE_DXNR = FloatMalloc (NearElectricCount);
  NE_DYNR = FloatMalloc (NearElectricCount);
  NE_DZNR = FloatMalloc (NearElectricCount);
 }
 for (i = 0; i < NearElectricCount; i++)
  fscanf (fp, "%d %d %d %d %f %f %f %f %f\n",
       &NE_NEAR[i], &NE_NRX[i], &NE_NRY[i], &NE_NRZ[i], &NE_XNR[i],
       &NE_YNR[i], &NE_ZNR[i], &NE_DXNR[i], &NE_DYNR[i], &NE_DZNR[i]);

 /* Read in Near Magnetic Field data */
 fscanf (fp, "Near Magnetic Field %d\n", &NearMagneticCount);
 XtFree ((char *) NH_NEAR);
 NH_NEAR = NULL;
 XtFree ((char *) NH_NRX);
 NH_NRX = NULL;
 XtFree ((char *) NH_NRY);
 NH_NRY = NULL;
 XtFree ((char *) NH_NRZ);
 NH_NRZ = NULL;
 XtFree ((char *) NH_XNR);
 NH_XNR = NULL;
 XtFree ((char *) NH_YNR);
 NH_YNR = NULL;
 XtFree ((char *) NH_ZNR);
 NH_ZNR = NULL;
 XtFree ((char *) NH_DXNR);
 NH_DXNR = NULL;
 XtFree ((char *) NH_DYNR);
 NH_DYNR = NULL;
 XtFree ((char *) NH_DZNR);
 NH_DZNR = NULL;
 if (NearMagneticCount > 0) {
  NH_NEAR = IntMalloc (NearMagneticCount);
  NH_NRX = IntMalloc (NearMagneticCount);
  NH_NRY = IntMalloc (NearMagneticCount);
  NH_NRZ = IntMalloc (NearMagneticCount);
  NH_XNR = FloatMalloc (NearMagneticCount);
  NH_YNR = FloatMalloc (NearMagneticCount);
  NH_ZNR = FloatMalloc (NearMagneticCount);
  NH_DXNR = FloatMalloc (NearMagneticCount);
  NH_DYNR = FloatMalloc (NearMagneticCount);
  NH_DZNR = FloatMalloc (NearMagneticCount);
 }
 for (i = 0; i < NearMagneticCount; i++)
  fscanf (fp, "%d %d %d %d %f %f %f %f %f\n",
       &NH_NEAR[i], &NH_NRX[i], &NH_NRY[i], &NH_NRZ[i], &NH_XNR[i],
       &NH_YNR[i], &NH_ZNR[i], &NH_DXNR[i], &NH_DYNR[i], &NH_DZNR[i]);

 /* Read in Radiation Pattern data */
 fscanf (fp, "Radiation Pattern %d\n", &RadiationPatternCount);
 XtFree ((char *) RP_I1);
 RP_I1 = NULL;
 XtFree ((char *) RP_NTH);
 RP_NTH = NULL;
 XtFree ((char *) RP_NPH);
 RP_NPH = NULL;
 XtFree ((char *) RP_XNDA);
 RP_XNDA = NULL;
 XtFree ((char *) RP_THETS);
 RP_THETS = NULL;
 XtFree ((char *) RP_PHIS);
 RP_PHIS = NULL;
 XtFree ((char *) RP_DTH);
 RP_DTH = NULL;
 XtFree ((char *) RP_DPH);
 RP_DPH = NULL;
 XtFree ((char *) RP_RFLD);
 RP_RFLD = NULL;
 XtFree ((char *) RP_GNOR);
 RP_GNOR = NULL;
 if (RadiationPatternCount > 0) {
  RP_I1 = IntMalloc (RadiationPatternCount);
  RP_NTH = IntMalloc (RadiationPatternCount);
  RP_NPH = IntMalloc (RadiationPatternCount);
  RP_XNDA = IntMalloc (RadiationPatternCount);
  RP_THETS = FloatMalloc (RadiationPatternCount);
  RP_PHIS = FloatMalloc (RadiationPatternCount);
  RP_DTH = FloatMalloc (RadiationPatternCount);
  RP_DPH = FloatMalloc (RadiationPatternCount);
  RP_RFLD = FloatMalloc (RadiationPatternCount);
  RP_GNOR = FloatMalloc (RadiationPatternCount);
 }
 for (i = 0; i < RadiationPatternCount; i++)
  fscanf (fp, "%d %d %d %d %f %f %f %f %f\n",
       &RP_I1[i], &RP_NTH[i], &RP_NPH[i], &RP_XNDA[i], &RP_THETS[i],
       &RP_PHIS[i], &RP_DTH[i], &RP_DPH[i], &RP_RFLD[i], &RP_GNOR[i]);


 fscanf (fp, "Print Options for Charge %d\n", &PrintChargeCount);
 XtFree ((char *) PQ_IPTFLQ);
 PQ_IPTFLQ = NULL;
 XtFree ((char *) PQ_IPTAQ);
```

```
  PQ_IPTAQ = NULL;
  XtFree ((char *) PQ_Distance1);
  PQ_Distance1 = NULL;
  XtFree ((char *) PQ_Distance2);
  PQ_Distance2 = NULL;
  if (PrintChargeCount > 0) {
    PQ_IPTFLQ = IntMalloc (PrintChargeCount);
    PQ_IPTAQ = IntMalloc (PrintChargeCount);
    PQ_Distance1 = FloatMalloc (PrintChargeCount);
    PQ_Distance2 = FloatMalloc (PrintChargeCount);
  }

  /* Read in Print Options data */
  for (i = 0; i < PrintChargeCount; i++)
    fscanf (fp, "%d %d %f %f\n", &PQ_IPTFLQ[i], &PQ_IPTAQ[i],
        &PQ_Distance1[i], &PQ_Distance2[i]);

  fscanf (fp, "Print Electrical Lengths %d\n", &PrintLengthCount);
  XtFree ((char *) PS_FLG);
  PS_FLG = NULL;
  if (PrintLengthCount > 0)
    PS_FLG = IntMalloc (PrintLengthCount);
  for (i = 0; i < PrintLengthCount; i++)
    fscanf (fp, "%d\n", &PS_FLG[i]);

  fscanf (fp, "Print Options for Current %d\n", &PrintCurrentCount);
  XtFree ((char *) PT_IPTFLQ);
  PT_IPTFLQ = NULL;
  XtFree ((char *) PT_IPTAQ);
  PT_IPTAQ = NULL;
  XtFree ((char *) PT_Distance1);
  PT_Distance1 = NULL;
  XtFree ((char *) PT_Distance2);
  PT_Distance2 = NULL;
  if (PrintCurrentCount > 0) {
    PT_IPTFLQ = IntMalloc (PrintCurrentCount);
    PT_IPTAQ = IntMalloc (PrintCurrentCount);
    PT_Distance1 = FloatMalloc (PrintCurrentCount);
    PT_Distance2 = FloatMalloc (PrintCurrentCount);
  }
  for (i = 0; i < PrintCurrentCount; i++)
    fscanf (fp, "%d %d %f %f\n", &PT_IPTFLQ[i], &PT_IPTAQ[i],
        &PT_Distance1[i], &PT_Distance2[i]);

  fscanf (fp, "Execute %d\n", &ExecuteCount);

  fclose (fp);
  return (1);                   /* return 1 for successful write */

}

/*************************************************************************
 * Opens "inputFilename" and writes NEEDS input data out to it.
 */
int writeInputToFile (filename)

  char *filename;
{
  FILE *fp;
  int i;

  if ((fp = efopen (filename, "w")) == NULL)
    return (0);                 /* return 0 for failed write */

  /* Write out environment, dimension & frequency unit data */
  fprintf (fp, "Environment %d\n", EnvIndex);
  fprintf (fp, "Dimension %d\n", DimIndex);
  fprintf (fp, "Freq Unit %d\n", FrequencyIndex);

  /* Write out Comments data */
  fprintf (fp, "Comments %d\n", CommentCount);
  if (CommentCount) {
    for (i = 0; i < CommentCount; i++) {
      fprintf(fp, "%s\n", CM[i].line);
    }
  }
  /* Write out Node Coordinates data */
  fprintf (fp, "\nNodes %d\n", NodeCount);

  /* Read data into arrays */
  for (i = 0; i < NodeCount; i++)
    fprintf (fp, "%g %g %g\n", X[i], Y[i], Z[i]);

  /* Write out Straight Wires data */
  fprintf (fp, "Straight Wires %d\n", SWireCount);
  for (i = 0; i < SWireCount; i++)
    fprintf (fp, "%d %d %d %d %g\n", GW_ITG[i], GW_END1[i], GW_END2[i],
        GW_NS[i], GW_RAD[i]);

  /* Write out Tapered Wires data */
  fprintf (fp, "Tapered Wires %d\n", TaperWireCount);
  for (i = 0; i < TaperWireCount; i++)
    fprintf (fp, "%d %d %d %d %d %g %g %g %g %g\n", GC_ITG[i], GC_END1[i],
```

```
        GC_END2[i], GC_NS[i], GC_IX[i], GC_RDEL[i], GC_RAD1[i],
        GC_RAD2[i], GC_DEL1[i], GC_DEL2[i]);

 /* Write out Cantenary Wires data */
 fprintf (fp, "Cantenary Wires %d\n", CWireCount);
 for (i = 0; i < CWireCount; i++)
  fprintf (fp, "%d %d %d %d %g %d %g %g\n", CW_ITG[i], CW_END1[i],
        CW_END2[i], CW_NS[i], CW_RAD[i], CW_ICAT[i], CW_RHM[i], CW_ZM[i]);

 /* Write out Wire Arc data */
 fprintf (fp, "Wire Arc %d\n", WireArcCount);
 for (i = 0; i < WireArcCount; i++)
  fprintf (fp, "%d %d %g %g %g %g\n", GA_ITG[i], GA_NS[i], GA_RADA[i],
        GA_ANG1[i], GA_ANG2[i], GA_RAD[i]);

 /* Write out Helix & Spiral data */
 fprintf (fp, "Helix/Spiral %d\n", HelixOrSpiralCount);
 for (i = 0; i < HelixOrSpiralCount; i++)
  fprintf (fp, "%d %d %g %g %g %g %g %g %g\n", GH_ITG[i],
        GH_NS[i], GH_TURNS[i], GH_ZLEN[i], GH_HR1[i], GH_HR2[i],
        GH_WR1[i], GH_WR2[i], GH_ISPX[i]);

 /* Write out Surface Patch data */
 fprintf (fp, "Surface Patch %d\n", SurfacePatchCount);
 for (i = 0; i < SurfacePatchCount; i++)
  fprintf (fp, "%d %d %d %d %d\n", SP_NS[i],
        SP_Corner1[i], SP_Corner2[i], SP_Corner3[i], SP_Corner4[i]);

 /* Write out Multiple Patches data */
 fprintf (fp, "Multiple Patch %d\n", MultiplePatchCount);
 for (i = 0; i < MultiplePatchCount; i++)
  fprintf (fp, "%d %d %d %d %d\n",
        SM_Corner1[i], SM_Corner2[i], SM_Corner3[i],
        SM_Number12[i], SM_Number23[i]);

 /* Write out Transformations data */
 fprintf (fp, "Transform %d\n", TransformCount);
 for (i = 0; i < TransformCount; i++)
  fprintf (fp, "%d %d %g %g %g %g %g %g\n",
        GM_ITGI[i], GM_NRPT[i], GM_ROX[i], GM_ROY[i], GM_ROZ[i], GM_XS[i],
        GM_YS[i], GM_ZS[i]);

 /* Write out Rotations data */
 fprintf (fp, "Rotation %d\n", RotationCount);
 for (i = 0; i < RotationCount; i++)
  fprintf (fp, "%d %d\n", GR_ITGI[i], GR_NR[i]);

 /* Write out Reflections data*/
 fprintf (fp, "Reflection %d\n", ReflectionCount);
 for (i = 0; i < ReflectionCount; i++)
  fprintf (fp, "%d %d\n", GX_ITGI[i], GX_IXYZ[i]);

 /* Write out Frequency data */
 fprintf (fp, "Frequency %d\n", FrequencyCount);
 for (i = 0; i < FrequencyCount; i++)
  fprintf (fp, "%d %d %g %g\n", FR_IFRQ[i], FR_NFRQ[i], FR_FMHZ[i],
        FR_DELFRQ[i]);

 /* Write out Impedance Loading data */
 fprintf (fp, "Load %d\n", LoadsCount);
 for (i = 0; i < LoadsCount; i++)
  fprintf (fp, "%d %d %g %g %g %g %g\n", LD_LDTYP[i], LD_Tag[i],
        LD_Distance[i], LD_Distance2[i], LD_ZLR[i], LD_ZLI[i], LD_ZLC[i]);

 /* Write out Voltage Sources data */
 fprintf (fp, "Voltage Sources %d\n", VoltageSourcesCount);
 for (i = 0; i < VoltageSourcesCount; i++)
  fprintf (fp, "%d %d %d %g %g %g %g %g\n", EX_Type[i], EX_Wire[i],
        EX_Format[i], EX_Distance[i], EX_Magnitude[i], EX_Phase[i],
        EX_Normal[i]);

 /* Write out Incident Plane Wave data */
 fprintf (fp, "Incident Plane Wave %d\n", IncidentPlaneWaveCount);
 for (i = 0; i < IncidentPlaneWaveCount; i++)
  fprintf (fp, "%d %d %d %g %g %g %g %g %g %g\n", EX_TYPE[i],
        EX_THETA_NUM[i], EX_PHI_NUM[i], EX_THETA_LO[i], EX_PHI_LO[i],
        EX_THETA_STEP [i], EX_PHI_STEP[i], EX_POL_ANGLE[i],
        EX_POL_RATIO[i], EX_INC_MAG[i]);

 /* Write out Transmission Line data */
 fprintf (fp, "Transmission Lines %d\n", TransmissionLinesCount);
 for (i = 0; i < TransmissionLinesCount; i++)
  fprintf (fp, "%d %d %g %g %g %g %g %g %g\n", TL_Wire1[i], TL_Wire2[i],
        TL_Distance1[i], TL_Distance2[i], TL_ZC[i], TL_TLEN[i], TL_Y1R[i],
        TL_Y1I[i], TL_Y2R[i], TL_Y2I[i]);

 /* Write out Two Port Networks data */
 fprintf (fp, "Two Port Nets %d\n", TwoPortNetsCount);
 for (i = 0; i < TwoPortNetsCount; i++)
  fprintf (fp, "%d %d %g %g %g %g %g %g %g\n", NT_Wire1[i], NT_Wire2[i],
        NT_Distance1[i], NT_Distance2[i], NT_Y11R[i], NT_Y11I[i],
        NT_Y12R[i], NT_Y12I[i], NT_Y22R[i], NT_Y22I[i]);
```

```
/* Write out Insulated Wires data */
fprintf (fp, "Insulated Wires %d\n", InsulatedWiresCount);
for (i = 0; i < InsulatedWiresCount; i++)
  fprintf (fp, "%d %d %g %g %g %g %g\n", IS_I1[i], IS_ITAG[i],
        IS_Distance1[i], IS_Distance2[i], IS_EPSR[i], IS_SIG[i],
        IS_RADI[i]);

/* Write out Ground Parameter data */
fprintf (fp, "Ground Parameters %d\n", GroundParamCount);
for (i = 0; i < GroundParamCount; i++)
  fprintf (fp, "%d %d %g %g %g %g %g %s\n",
        GN_IPERF[i], GN_NRADL[i], GN_EPSR[i], GN_SIG[i],
        GN_F3[i], GN_F4[i], GN_F5[i], GN_F6[i], GN_Filename);

/* Write out Additional Ground Parameters data */
fprintf (fp, "Addition Ground Parameters %d\n", AddGroundParamCount);
for (i = 0; i < AddGroundParamCount; i++)
  fprintf (fp, "%d %g %g %g %g\n", GD_ICLIF[i], GD_EPSR2[i], GD_SIG2[i],
        GD_CLT[i], GD_CHT[i]);

/* Upper-Medium Parameters */
fprintf (fp, "Upper-Medium Parameters %d\n", UpperMediumParamCount);
for (i = 0; i < UpperMediumParamCount; i++)
  fprintf (fp, "%g %g\n", UM_EPSR[i], UM_SIG[i]);

/* Write out Maximum Coupling Calculation data */
fprintf (fp, "Max Coupling %d\n", MaxCouplingCount);
for (i = 0; i < MaxCouplingCount; i++)
  fprintf (fp, "%d %d %g %g\n", CP_TAG1[i], CP_TAG2[i], CP_Distance1[i],
        CP_Distance2[i]);

/* Write out Near Electric Field data */
fprintf (fp, "Near Electric Field %d\n", NearElectricCount);
for (i = 0; i < NearElectricCount; i++)
  fprintf (fp, "%d %d %d %d %g %g %g %g %g\n",
        NE_NEAR[i], NE_NRX[i], NE_NRY[i], NE_NRZ[i], NE_XNR[i], NE_YNR[i],
        NE_ZNR[i], NE_DXNR[i], NE_DYNR[i], NE_DZNR[i]);

/* Write out Near Magnetic Field data */
fprintf (fp, "Near Magnetic Field %d\n", NearMagneticCount);
for (i = 0; i < NearMagneticCount; i++)
  fprintf (fp, "%d %d %d %d %g %g %g %g %g %g\n",
        NH_NEAR[i], NH_NRX[i], NH_NRY[i], NH_NRZ[i], NH_XNR[i], NH_YNR[i],
        NH_ZNR[i], NH_DXNR[i], NH_DYNR[i], NH_DZNR[i]);

/* Write out Radiation Pattern data */
fprintf (fp, "Radiation Pattern %d\n", RadiationPatternCount);
for (i = 0; i < RadiationPatternCount; i++)
  fprintf (fp, "%d %d %d %d %g %g %g %g %g %g\n",
        RP_I1[i], RP_NTH[i], RP_NPH[i], RP_XNDA[i], RP_THETS[i],
        RP_PHIS[i], RP_DTH[i], RP_DPH[i], RP_RFLD[i], RP_GNOR[i]);

/* Write out Print Options data */
fprintf (fp, "Print Options for Charge %d\n", PrintChargeCount);
for (i = 0; i < PrintChargeCount; i++)
  fprintf (fp, "%d %d %g %g\n",
    PQ_IPTFLQ[i], PQ_IPTAQ[i], PQ_Distance1[i], PQ_Distance2[i]);
fprintf (fp, "Print Electrical Lengths %d\n", PrintLengthCount);
for (i = 0; i < PrintLengthCount; i++)
  fprintf (fp, "%d\n", PS_FLG[i]);
fprintf (fp, "Print Options for Current %d\n", PrintCurrentCount);
for (i = 0; i < PrintCurrentCount; i++)
  fprintf (fp, "%d %d %g %g\n",
    PT_IPTFLQ[i], PT_IPTAQ[i], PT_Distance1[i], PT_Distance2[i]);

fprintf (fp, "Execute %d\n", ExecuteCount);

fclose (fp);
saveAlert = False;          /* Reset saveAlert flag */
return (1);                 /* return 1 for successful write */

}

/*****************************************************************
 * Clears out old data
 */
void clearOldData ()
{
 /* Clear out element counts for all data arrays */
 NodeCount = 0;
 SWireCount = 0;
 TaperWireCount = 0;
 CWireCount = 0;
 WireArcCount = 0;
 HelixOrSpiralCount = 0;
 SurfacePatchCount = 0;
 MultiplePatchCount = 0;
 RotationCount = 0;
 ReflectionCount = 0;
 TransformCount = 0;
 FrequencyCount = 1;  /* Always have at least one frequency! */
 InsulatedWiresCount = 0;
 LoadsCount = 0;
```

```
  UpperMediumParamCount = 0;
  VoltageSourcesCount = 0;
  IncidentPlaneWaveCount = 0;
  TwoPortNetsCount = 0;
  TransmissionLinesCount = 0;
  MaxCouplingCount = 0;
  AddGroundParamCount = 0;
  NearElectricCount = 0;
  NearMagneticCount = 0;
  RadiationPatternCount = 0;
  GroundParamCount = 1;
  PrintChargeCount = 0;
  PrintCurrentCount = 0;
  PrintLengthCount = 0;
  ExecuteCount = 1;

  /* Set environment, dimension & frequency unit to defaults */
  FrequencyIndex = MHZ;
  EnvIndex = FREE_SPACE;
  DimIndex = METERS;

  /* Free Comments */
  XtFree ((char *)CM);
  CM = NULL;

  /* Set Frequency to defaults */
  FR_IFRQ = IntMalloc (FrequencyCount);
  FR_NFRQ = IntMalloc (FrequencyCount);
  FR_FMHZ = FloatMalloc (FrequencyCount);
  FR_DELFRQ = FloatMalloc (FrequencyCount);
  FR_IFRQ[0] = 0;
  FR_NFRQ[0] = 1;
  FR_FMHZ[0] = 299.8;
  FR_DELFRQ[0] = 0;

  /* Set Ground Parameters to defaults */
  GN_IPERF = IntMalloc (GroundParamCount);
  GN_IPERF[0] = -1;
  XtFree (GN_Filename);
  GN_Filename = NULL;

} /* end clearOldData */

/*****************************************************************/
Boolean searchFile(filename)
char *filename;
{
  Widget w;
  char *ext, mask[10];
  XmString dirmask, file;
  XmStringTable fileList;
  Arg args[10];
  int i, fileCount = 0, n = 0;

  if (filename == NULL)
    return FALSE;

  w = XmCreateFileSelectionDialog(topLevel, "file",
              NULL, 0);
  ext = strrchr(filename, '.');
  if (ext == NULL)
    return FALSE;
  sprintf(mask, "*%s", ext);
  dirmask = XmStringLtoRCreate (mask, XmSTRING_DEFAULT_CHARSET);
  XmFileSelectionDoSearch(w, dirmask);
  XmStringFree (dirmask);
  XtSetArg (args [n], XmNfileListItems, &fileList); n++;
  XtSetArg (args [n], XmNfileListItemCount, &fileCount); n++;
  XtGetValues (w, args, n);
  file = XmStringLtoRCreate (filename, XmSTRING_DEFAULT_CHARSET);
  for (i = 0; i < fileCount; i++)
    if (XmStringCompare (file, fileList [i]))
      break;
  XmStringFree (file);
  XtDestroyWidget(XtParent(w));
  if (i < fileCount)
    return TRUE;
  else
    return FALSE;
} /* end searchFile */

/*****************************************************************/
void newFileAction()
{
  extern void createNewSaveDialog();
  char title[80];
  char *msg = "Input data have been modified.\nSave NEC file (*.nec) ?";

  closeAll();
  if (saveAlert)
    createNewSaveDialog(topLevel, "Warning", msg);
  else {
    clearDataInputs();
```

```
    sprintf(title, "NEEDS %s - [%s]", VERSION, "New File");
    XtVaSetValues(topLevel, XmNtitle, title, NULL);
  }

} /* end newFileAction */
```

cFileMenu.h:

```
#include "ctrlgeo.h"

/*
 * The following are the global variables for the NEEDS data set
 */

extern int  EnvIndex,          /* Environment array index */
        DimIndex,          /* Dimension array index */
        FrequencyIndex,    /* Frequency array index */
        NodeCount,         /* Node Coordinates */
        SWireCount,        /* Straight Wires */
        TaperWireCount,    /* Tapered Wires */
        CWireCount,        /* Catenary Wires */
        WireArcCount,      /* Wire Arcs */
        HelixOrSpiralCount,  /* Helix or Spiral Wires */
        SurfacePatchCount,   /* Surface Patches */
        MultiplePatchCount,  /* Multiple Patch Surface */
        RotationCount,     /* Rotation */
        ReflectionCount,   /* Reflections */
        TransformCount,    /* Transformations */
        FrequencyCount,    /* Frequencies */
        InsulatedWiresCount,  /* Insulated Wires */
        LoadsCount,        /* Impedance Loads */
        UpperMediumParamCount, /* Upper-Medium Parameters */
        VoltageSourcesCount,  /* Voltage Sources */
        IncidentPlaneWaveCount, /* Incident Plane Waves */
        TwoPortNetsCount,     /* 2-port non-radiating networks */
        TransmissionLinesCount, /* Transmission Lines */
        MaxCouplingCount,     /* Maximum Coupling Calculation */
        AddGroundParamCount,  /* Additional Ground Parameters */
        NearElectricCount,    /* Near Electric Field */
        NearMagneticCount,    /* Near Magnetic Field */
        RadiationPatternCount, /* Radiation Pattern */
        GroundParamCount,  /* Ground Parameters */
        PrintChargeCount,  /* Print Option for Charge */
        PrintCurrentCount,  /* Print Option for Current */
        PrintLengthCount,  /* Print Length of Segments */
            ExecuteCount,
        CommentCount;   /* lines of comments */

/* Comments data */
extern stringType *CM;

/* Node Coordinates data */

extern float  *X,              /* x coordinate */
          *Y,              /* y coordinate */
          *Z;              /* z coordinate */

/* added by russelll 6/15/95 */
/* Diagnostic error and warning definitions */
#define SegLen2WaveLenWarning      (1lu << 0)
#define SegLen2RadiusWarning       (1lu << 1)
#define Radius2WaveLenWarning      (1lu << 2)
#define JunctionSegLenRatioWarning (1lu << 3)
#define JunctionRadiusRatioWarning (1lu << 4)
#define SegLen2WaveLenError        (1lu << 5)
#define SegLen2RadiusError         (1lu << 6)
#define Radius2WaveLenError        (1lu << 7)
#define CoincidentWireError        (1lu << 8)
#define JunctionSegLenRatioError   (1lu << 9)
#define JunctionRadiusRatioError   (1lu << 10)
#define JunctionMatchPointError    (1lu << 11)
#define CrossedWireError (1lu << 12)
#define InvalidSheathRadiusError   (1lu << 13)

/* Global variable for indicating the errors and warnings of wires */
extern unsigned long *wireErrors;

/* Straight Wires data */

extern int   *GW_ITG,          /* tag of wire */
         *GW_END1,         /* node of end 1 */
         *GW_END2,         /* node of end 2 */
         *GW_NS;           /* number of segments */
extern float *GW_RAD;          /* radius */

/* Tapered Wires data */

extern int   *GC_ITG,          /* tag of wire */
         *GC_END1,         /* node of end 1 */
         *GC_END2,         /* node of end 2 */
         *GC_NS,           /* number of segments */
         *GC_IX;           /* type of taper */
```

```
extern float  *GC_RDEL,        /* ratio of segment lengths */
        *GC_RAD1,       /* radius of 1st segment */
        *GC_RAD2,       /* radius of 2nd segment */
        *GC_DEL1,       /* length of 1st segment */
        *GC_DEL2;       /* length of 2nd segment */

/* Catenary Wires data */

extern int   *CW_ITG,          /* tag of wire */
        *CW_END1,       /* node of end 1 */
        *CW_END2,       /* node of end 2 */
        *CW_ICAT,       /* type of catenary */
        *CW_NS;         /* number of segments */
extern float  *CW_RAD,         /* radius */
        *CW_RHM,        /* distance */
        *CW_ZM;         /* height */

/* Wire Arc data */

extern int   *GA_ITG,          /* tag */
        *GA_NS;         /* number of segments */
extern float  *GA_RADA,        /* Arc radius */
        *GA_ANG1,       /* angle of 1st end */
        *GA_ANG2,       /* angle of 2nd end */
        *GA_RAD;        /* Wire Radius */

/* Helix or Spiral data */

extern int   *GH_ITG,          /* tag */
        *GH_NS;         /* number of segments */
extern float  *GH_TURNS,       /* number of turns in spiral */
        *GH_ZLEN,       /* length of spiral of heliz along z-axis */
        *GH_HR1,        /* radius of spiral at starting end */
        *GH_HR2,        /* radius of sprial at final end */
        *GH_WR1,        /* radius of wire at starting end */
        *GH_WR2,        /* radius of wire at final end */
        *GH_ISPX;       /* 0=log spiral, 1=Archimedes spiral */

/* Multiple Patches data */

extern int   *SM_Corner1,      /* Node of corner1 of quadrangle */
        *SM_Corner2,    /* Node of corner2 of quadrangle */
        *SM_Corner3,    /* Node of corner3 of quadrangle */
        *SM_Number12,   /* Number of patches from 1 to 2 */
        *SM_Number23;   /* Number of patches from 2 to 3 */

/* Surface Patches data */

extern int   *SP_NS,           /* Patch type */
        *SP_Corner1,    /* Node of corner1 of quadrangle */
        *SP_Corner2,    /* Node of corner2 of quadrangle */
        *SP_Corner3,    /* Node of corner3 of quadrangle */
        *SP_Corner4;    /* Node of corner4 of quadrangle */

/* Rotations */

extern int   *GR_ITGI,         /* Tag number increment */
        *GR_NR;         /* # of times in array */

/* Reflections */

extern int   *GX_ITGI,         /* Tag number increment */
        *GX_IXYZ;       /* Reflection control */

/* Transformations */

extern int   *GM_ITGI,         /* Tag number increment */
        *GM_NRPT;       /* # of new structures to generate */
extern float  *GM_ROX,         /* x-angle rotation */
        *GM_ROY,        /* y-angle rotation */
        *GM_ROZ,        /* z-angle rotation */
        *GM_XS,         /* shift in x direction */
        *GM_YS,         /* shift in y direction */
        *GM_ZS;         /* shift in z direction */

/* Frequency data */

extern int   *FR_IFRQ,         /* Type of frequency stepping */
        *FR_NFRQ;       /* Number of frequency steps */
extern float  *FR_FMHZ,        /* Frequency of MHz or starting freq in range */
        *FR_DELFRQ;     /* Frequency stepping increment */

/* Ground Parameters */
extern int   *GN_IPERF,        /* Ground type flag */
        *GN_NRADL;      /* Number of radial wires */
extern float  *GN_EPSR,        /* Relative dielectric constant */
        *GN_SIG,        /* Conductivity in S/m of the ground */
        *GN_F3,
        *GN_F4,
        *GN_F5,
        *GN_F6;
extern char  *GN_Filename;
```

```
/* Insulating sheath of dielectric or lossy material on a wire */

extern int   *IS_I1,        /* -1 to cancel, 0 for new data */
        *IS_ITAG;      /* Identified wire for insulation */
extern float *IS_Distance1,   /* Distance along wire for insulation start */
        *IS_Distance2,   /* Distance along wire for insulation end */
        *IS_EPSR,       /* Relative permittivity of sheath material */
        *IS_SIG,       /* Conductivity of sheath material */
        *IS_RADI;      /* Radius of sheath */

/* Impedance Loading */

extern int   *LD_LDTYP,         /* Type of loading used */
        *LD_Tag;          /* Wire on which load is located */
extern float *LD_Distance,      /* Distance of load from end #1 on wire */
        *LD_Distance2,      /* End distance */
        *LD_ZLR,          /* Resistance */
        *LD_ZLI,          /* Inductance */
        *LD_ZLC;          /* Capacitance */

/* Upper-Medium Parameters */

extern float *UM_EPSR,          /* Relative permittivity of medium */
        *UM_SIG;          /* Conductivity of medium in S/m */

/* Voltage Sources */

extern int   *EX_Type,       /* Type of excitation that is used */
        *EX_Wire,       /* Wire on which voltage source is located */
        *EX_Format;     /* 1 to request input impedance */
extern float *EX_Distance,    /* Distance of voltage source from end #1 */
        *EX_Magnitude,   /* Magnitude of voltage source in volts */
        *EX_Phase,      /* Phase of voltage source in degrees */
        *EX_Normal;     /* Normalization factor for impedance */

/* Incident Plane Waves */

extern int   *EX_TYPE,       /* Type of excitation that is used */
        *EX_THETA_NUM,   /* Number of Theta angles */
        *EX_PHI_NUM;     /* Number of Phi angles */
extern float *EX_THETA_LO,    /* Angle Theta */
        *EX_PHI_LO,      /* Angle Phi */
        *EX_THETA_STEP,  /* Stepping increment for Theta */
        *EX_PHI_STEP,    /* Stepping increment for Phi */
        *EX_POL_ANGLE,   /* Polarization angle */
        *EX_POL_RATIO,   /* Ratio of minor axis to major axis */
        *EX_INC_MAG;     /* Magnitude of electric field */

/* Two-port non-radiating networks */

extern int   *NT_Wire1,      /* Wire for first end of network */
        *NT_Wire2;      /* Wire for second end of network */
extern float *NT_Distance1,   /* Distance along wire 1 for location */
        *NT_Distance2,   /* Distance along wire 2 for location */
        *NT_Y11R,       /* Real part of Y11 in mhos */
        *NT_Y11I,       /* Imaginary part of Y11 in mhos */
        *NT_Y12R,       /* Real part of Y12 in mhos */
        *NT_Y12I,       /* Imaginary part of Y12 in mhos */
        *NT_Y22R,       /* Real part of Y22 in mhos */
        *NT_Y22I;       /* Imaginary part of Y22 in mhos */

/* Transmission Lines */

extern int   *TL_Wire1,      /* Wire for 1st end of transmission line */
        *TL_Wire2;      /* Wire for 2nd end of transmission line */
extern float *TL_Distance1,   /* Distance along wire 1 for location */
        *TL_Distance2,   /* Distance along wire 2 for location */
        *TL_ZC,         /* Characteristic impedance in ohms */
        *TL_TLEN,       /* Length of transmission line in meters */
        *TL_Y1R,        /* Real part of shunt admittance across end1*/
        *TL_Y1I,        /* Imaginary part of shunt admittance */
        *TL_Y2R,        /* Real part of shunt admittance across end2*/
        *TL_Y2I;        /* Imaginary part of shunt admittance */

/* Maximum Coupling Calculation */

extern int   *CP_TAG1,       /* Tag number of 1st wire segment */
        *CP_TAG2;       /* Tag number of 2nd wire segment */
extern float *CP_Distance1,   /* Distance along wire 1 for location */
        *CP_Distance2;   /* Distance along wire 2 for location */

/* Additional Ground Parameters */

extern int   *GD_ICLIF;      /* Boundary type */
extern float *GD_EPSR2,       /* Relative dielectric of 2nd medium */
        *GD_SIG2,       /* Conductivity of 2nd medium in S/m */
        *GD_CLT,        /* Distance from origin to boundary */
        *GD_CHT;        /* Distance from medium 2 to medium 1 */

/* Near Electric Field */

extern int   *NE_NEAR,       /* Coordinate type */
        *NE_NRX,        /* Number of points in x direction */
```

```
        *NE_NRY,        /* Number of points in y direction */
        *NE_NRZ;        /* Number of points in z direction */
extern float  *NE_XNR,          /* Initial x coordinate */
        *NE_YNR,        /* Initial y coordinate */
        *NE_ZNR,        /* Initial z coordinate */
        *NE_DXNR,        /* Increment for x */
        *NE_DYNR,        /* Increment for y */
        *NE_DZNR;        /* Increment for z */

/* Near Magnetic Field */

extern int    *NH_NEAR,         /* Coordinate type */
        *NH_NRX,        /* Number of points in x direction */
        *NH_NRY,        /* Number of points in y direction */
        *NH_NRZ;        /* Number of points in z direction */
extern float  *NH_XNR,          /* Initial x coordinate */
        *NH_YNR,        /* Initial y coordinate */
        *NH_ZNR,        /* Initial z coordinate */
        *NH_DXNR,        /* Increment for x */
        *NH_DYNR,        /* Increment for y */
        *NH_DZNR;        /* Increment for z */

/* Print Options */

extern int    *PQ_IPTFLQ,       /* Options for printing charge densities */
        *PQ_IPTAQ;        /* Tag number of Wire to be printed */
extern float  *PQ_Distance1,    /* Beginning location for wire */
        *PQ_Distance2;    /* End location for wire */
extern int    *PS_FLG,          /* Print electrical lengths of segments */
        *PT_IPTFLQ,       /* Print currents on wire segments */
        *PT_IPTAQ;        /* Tag number of Wire to be printed */
extern float  *PT_Distance1,    /* Beginning location for wire */
        *PT_Distance2;    /* End location for wire */

/* Radiation Pattern */

extern int    *RP_I1,           /* Mode of calculation */
        *RP_NTH,        /* Number of theta values */
        *RP_NPH,        /* Number of phi values */
        *RP_XNDA;       /* Gain options */
extern float  *RP_THETS,        /* Initial theta */
        *RP_PHIS,       /* Initial phi */
        *RP_DTH,        /* Increment for theta */
        *RP_DPH,        /* Increment for phi */
        *RP_RFLD,       /* Radial distance */
        *RP_GNOR;       /* Gain normalization factor */
```

A.6 cNodeCoord.c, fNodeCoord.c

cNodeCoord.c:

```
/*
 * Callbacks & procedures for the Node Coordinates form
 */
#include <stdio.h>
#include <stdlib.h>
#include <X11/IntrinsicP.h>
#include <X11/ShellP.h>
#include <Xm/List.h>
#include <Xm/MessageB.h>
#include <Xm/SelectioB.h>
#include <Xm/Text.h>
#include "control.h"

extern Widget topLevel;
extern Widget nodeCoordShell;
extern Widget *nodeCoordEnvironment, *nodeCoordDimension, nodeCoordList;

/* variables for holding Node Coordinates data */
float *X, *Y, *Z;
int NodeCount = 0;
int EnvIndex = FREE_SPACE;
int DimIndex = METERS;

extern void createNodeCoordWindow ();
extern int getListPosition ();
extern int getListCount ();
extern void modifyNodeCoordListAll ();

static void clearXYZtextFields ();
static XmString *createNodeCoordStringTable ();
static XmString createXmStringFromXYZ ();
static void getXYZtextFields ();
static void resetListCount ();
static void setEditButtonState ();

/*****************************************************************************/
void openNodeCoordWindow ()

{
 XmString *stringTable;
 Arg args [2];
 Widget menu;
 int i;
 ShellWidget sw = (ShellWidget) nodeCoordShell;

 if (nodeCoordShell == NULL)
   createNodeCoordWindow ();
 else if (sw->shell.popped_up)
   return; /* Don't do anything if already open */

 /* Create string table & update list */
 stringTable = createNodeCoordStringTable ();
 XtSetArg (args [0], XmNitems, stringTable);
 XtSetArg (args [1], XmNitemCount, NodeCount);
 XtSetValues (nodeCoordList, args, 2);

 for (i = 0; i < NodeCount; i++)
   XmStringFree (stringTable[i]);
 XtFree ((char *) stringTable);

 /* Update the Environment option */
 menu = XtNameToWidget (nodeCoordShell, "nodeCoordForm.frame1.Environment");
 XtSetArg (args [0], XmNmenuHistory, nodeCoordEnvironment[EnvIndex]);
 XtSetValues (menu, args, 1);

 /* Update the Dimension option */
 menu = XtNameToWidget (nodeCoordShell, "nodeCoordForm.frame2.Dimension");
 XtSetArg (args [0], XmNmenuHistory, nodeCoordDimension[DimIndex]);
 XtSetValues (menu, args, 1);

 XtPopup (nodeCoordShell, XtGrabNone);
 setEditButtonState ();

} /* end openNodeCoordWindow */

/*****************************************************************************/
static XmString *createNodeCoordStringTable ()
{
 XmString *stringTable;
 char string [132];
 int i;

 if (NodeCount > 0) {

   /* Allocate memory for string table */
   stringTable = (XmString *) XtMalloc (sizeof (XmString) * NodeCount);
```

```
    /* Create strings to be placed in string table */
    for (i = 0; i < NodeCount; i++){
      sprintf (string, "%-20d%12g%12g%12g", i+1, X[i], Y[i], Z[i]);
      stringTable[i] = XmStringCreateSimple (string);
    }
  } else
    stringTable = NULL;
  return (stringTable);
} /* end createNodeCoordStringTable */

/*****************************************************************/

void nodeCoordListCB (w, clientData, cb)
    Widget w;
    XtPointer clientData;
    XmListCallbackStruct *cb;
{
  char x [15], y [15], z [15], node [5];
  char *string;
  Widget xWidget, yWidget, zWidget;

  if (cb->selected_item_count == 1) {
    XmStringGetLtoR (cb->selected_items[0], XmSTRING_DEFAULT_CHARSET, &string);

    /* Get text field widgets */
    getXYZtextFields (&xWidget, &yWidget, &zWidget);

    /* Get data & put into text fields */
    sscanf (string, "%s %s %s %s", node, x, y, z);
    XmTextSetString (xWidget, x);
    XmTextSetString (yWidget, y);
    XmTextSetString (zWidget, z);
    XtFree ((char *) string);
  }

  /* Enable edit buttons */
  setEditButtonState ();

  w = w; /* Make compiler happy */
  clientData = clientData;

} /* end nodeCoordListCB */

/*****************************************************************/

void nodeTextCB (w)
    Widget w;
{
  char *text;
  int nodeIndex;

  text = XmTextGetString (w);
  if (nodeIndex = atoi (text))
    XmListSelectPos (nodeCoordList, nodeIndex, TRUE);
  XtFree (text);
} /* end nodeTextCB */

/*****************************************************************/

static void setEditButtonState ()
{
  Widget addButton, modifyButton, deleteButton, xText, yText, zText;
  int count;

  /* Get the add, modify & delete buttons. */
  modifyButton = XtNameToWidget (nodeCoordShell,
     "nodeCoordForm.nodeCoordBox.workArea.modifyButton");
  deleteButton = XtNameToWidget (nodeCoordShell,
     "nodeCoordForm.nodeCoordBox.workArea.deleteButton");
  addButton = XtNameToWidget (nodeCoordShell,
     "nodeCoordForm.nodeCoordBox.workArea.addButton");

  count = getListCount (nodeCoordList);
  if (count == 1) {
    XtSetSensitive (addButton, TRUE);
    XtSetSensitive (modifyButton, TRUE);
    XtSetSensitive (deleteButton, TRUE);
  } else {
    getXYZtextFields (&xText, &yText, &zText);
    XmTextSetString (xText, "");
    XmTextSetString (yText, "");
    XmTextSetString (zText, "");
    if (count > 1) {
      XtSetSensitive (addButton, FALSE);
      XtSetSensitive (modifyButton, TRUE);
      XtSetSensitive (deleteButton, TRUE);
    }
    else {
      XtSetSensitive (addButton, TRUE);
      XtSetSensitive (modifyButton, FALSE);
      XtSetSensitive (deleteButton, FALSE);
    }
  }
```

```
} /* end setEditButtonState */

/*****************************************************************************/
static void getXYZtextFields (xText, yText, zText)
    Widget  *xText, *yText, *zText;
{
  /* Get the x, y, & z text fields */
  *xText = XtNameToWidget (nodeCoordShell,
     "nodeCoordForm.nodeCoordBox.workArea.xText");
  *yText = XtNameToWidget (nodeCoordShell,
     "nodeCoordForm.nodeCoordBox.workArea.yText");
  *zText = XtNameToWidget (nodeCoordShell,
     "nodeCoordForm.nodeCoordBox.workArea.zText");
} /* end getXYZtextFields */

/*****************************************************************************/
static XmString createXmStringFromXYZ (nodeIndex)
    int nodeIndex;
{
  Widget xText, yText, zText, button, env;
  char string [80], *x, *y, *z;
  float fz;
  Arg args [1];

  getXYZtextFields (&xText, &yText, &zText);

  /* Get the environment type. We need to check z-value if the
   * environment is Ground Plane.
   */
  button = XtNameToWidget (nodeCoordShell,
                "nodeCoordForm.frame1.Environment");
  XtSetArg (args [0], XmNmenuHistory, &env);
  XtGetValues (button, args, 1);

  /* Create a string using inputs */
  x = XmTextGetString (xText);
  y = XmTextGetString (yText);
  z = XmTextGetString (zText);
  fz = atof (z);

  /* Sound bell and set z to 0 if environment is Ground Plane and
   * z < 0.
   */
  if ((strcmp ("Ground Plane", XtName (env)) == 0) && (fz < 0)) {
    fz = 0.0;
    XBell (XtDisplay (xText), 50);
  }
  sprintf (string, "%-20d%12g%12g%12g", nodeIndex, atof(x), atof(y), fz);
  XtFree ((char *) x);
  XtFree ((char *) y);
  XtFree ((char *) z);
  return (XmStringCreateSimple (string));

} /* end createXmStringFromXYZ */

/*****************************************************************************/
void nodeCoordAddButtonCB (void)
{
  int nodeIndex;
  XmString xmString;
  Widget text;

  /* Get the current list selection */
  nodeIndex = getListPosition (nodeCoordList) + 1;

  if (nodeIndex == 1) nodeIndex = 0;

  /* Insert string into list */
  xmString = createXmStringFromXYZ (nodeIndex);
  XmListAddItem (nodeCoordList, xmString, nodeIndex);
  XmListSelectPos (nodeCoordList, nodeIndex, TRUE);
  XmStringFree (xmString);

  /* Reset the count */
  resetListCount (nodeIndex + 1);

  /* Move focues to x Text widget */
  text = XtNameToWidget (nodeCoordShell,
     "nodeCoordForm.nodeCoordBox.workArea.xText");

  /* Move focus to x Text widget */
  XtSetKeyboardFocus (nodeCoordShell, text);

  XmListSetPos(nodeCoordList, nodeIndex);

} /* end nodeCoordAddButtonCB */

/*****************************************************************************
 * Resets the list count from "begin" through the end of the list.
 */
static void resetListCount (begin)
    int begin;
{
```

```
  Arg args [2];
  int i, j, end, count;
  XmString *items, *newItems;
  char *string, newString [80];
  float x, y, z;

  /* Get list data */
  XtSetArg (args [0], XmNitemCount, &end);
  XtSetArg (args [1], XmNitems, &items);
  XtGetValues (nodeCoordList, args, 2);

  if (end > 0) {

   /* Allocate memory for string table */
   newItems = (XmString *) XtMalloc (sizeof (XmString) * (end - begin + 1));

   for (i=begin-1, j=0; i<end; i++, j++) {
    XmStringGetLtoR (items[i], XmSTRING_DEFAULT_CHARSET, &string);
    sscanf (string, "%d %f %f %f", &count, &x, &y, &z);
    sprintf (newString, "%-20d%12g%12g%12g", i + 1, x, y, z);
    newItems [j] = XmStringCreateSimple (newString);
    XtFree ((char *) string);
   }
   XmListReplaceItemsPos (nodeCoordList, newItems, j, begin);

   for (i = 0; i < j; i++)
    XmStringFree (newItems[i]);
   XtFree ((char *) newItems);
  }
 } /* end resetListCount */

 /*****************************************************************************/
 void nodeCoordModifyButtonCB (void)
 {
  int nodeIndex;
  XmString xmString;

  if (getListCount (nodeCoordList) > 1) {
   modifyNodeCoordListAll (nodeCoordList);
   setEditButtonState ();
   return;
  }

  /* Get the current list selection */
  if (nodeIndex = getListPosition (nodeCoordList)) {

   /* Replace string in list */
   xmString = createXmStringFromXYZ (nodeIndex);
   XmListReplaceItemsPos (nodeCoordList, &xmString, 1, nodeIndex);
   XmListSelectPos (nodeCoordList, nodeIndex, TRUE);
   XmStringFree (xmString);
  }
 } /* end nodeCoordModifyButton */

 /*****************************************************************************/
 void nodeCoordDeleteButtonCB (void)
 {
  int nodeIndex;
  int *positionList, count;
  int total;
  char msg[80];
  XEvent event;
  XtAppContext cxt = XtWidgetToApplicationContext(topLevel);
  Boolean sufficientNodeCount();

  /* Clear the text fields */
  clearXYZtextFields ();

  /* Get the current list selection */
  if (nodeIndex = getListPosition (nodeCoordList)) {

   /* Delete string in list
    */
   XmListGetSelectedPos (nodeCoordList, &positionList, &count);
   total = NodeCount - count;
   if (!sufficientNodeCount(total)) {
    sprintf(msg, "Insufficient number of nodes defined.\nCannot delete.");
    createMessageDialog(topLevel, "Warning", msg, XmDIALOG_WARNING);
    while (True) {
     XtAppNextEvent(cxt, &event);
     XtDispatchEvent(&event);
     if (event.xfocus.type == FocusOut) {
      break;
     }
    }
    XtFree ((char *)positionList);
    XmListDeselectAllItems(nodeCoordList);
    setEditButtonState ();
    return;
   }
   XmListDeletePositions (nodeCoordList, positionList, count);
   XtFree ((char *)positionList);
   resetListCount (nodeIndex);
```

```
    setEditButtonState ();
    XmListSelectPos (nodeCoordList, nodeIndex, TRUE);
    if (!XmListPosSelected(nodeCoordList, nodeIndex))
      XmListSelectPos (nodeCoordList, nodeIndex - 1, TRUE);
  }
} /* end nodeCoordDeleteButton */

/*****************************************************************/
void nodeCoordTextCB (w, nextText)
    Widget w, nextText;
{
  XtSetKeyboardFocus (nodeCoordShell, nextText);

  w = w;  /* Make compiler happy */

} /*end nodeCoordTextCB */

/*****************************************************************
 * Save the changes and close the window.
 */
void nodeCoordOkButtonCB (void)
{
  extern void nodeCoordApplyButtonCB ();

  nodeCoordApplyButtonCB ();
  XtPopdown (nodeCoordShell);

} /* end nodeCoordOkButtonCB */

/*****************************************************************
 * Save the changes and leave window open.
 */
void nodeCoordApplyButtonCB (void)
{
  Widget button, env, dim;
  XmString *items;
  Arg args [10];
  int n, count, i, node;
  float x, y, z;
  char *string;
  extern Boolean saveAlert;
  extern Boolean DRAW;

  n = 0;
  XtSetArg (args [n], XmNitems, &items); n++;
  XtSetArg (args [n], XmNitemCount, &count); n++;
  XtGetValues (nodeCoordList, args, n);

  /* Allocate X, Y, and Z arrays */
  X = (float *) XtRealloc ((char *) X, sizeof (float) * count);
  Y = (float *) XtRealloc ((char *) Y, sizeof (float) * count);
  Z = (float *) XtRealloc ((char *) Z, sizeof (float) * count);

  NodeCount = count;
  for (i=0; i<count; i++) {
    XmStringGetLtoR (items[i], XmSTRING_DEFAULT_CHARSET, &string);
    sscanf (string, "%d %f %f %f", &node, &x, &y, &z);
    X [i] = x;
    Y [i] = y;
    Z [i] = z;
    XtFree ((char *) string);
  }

  /* Save the environment & dimension */
  button = XtNameToWidget (nodeCoordShell,
                "nodeCoordForm.frame1.Environment");
  n = 0;
  XtSetArg (args [n], XmNmenuHistory, &env); n++;
  XtGetValues (button, args, n);
  saveEnvironment (env);
  button = XtNameToWidget (nodeCoordShell, "nodeCoordForm.frame2.Dimension");
  n = 0;
  XtSetArg (args [n], XmNmenuHistory, &dim); n++;
  XtGetValues (button, args, n);
  saveDimension (dim);

  saveAlert = True;
  DRAW = True;

} /* end nodeCoordApplyButtonCB */

/*****************************************************************
 * Ignore all changes and reset the values.
 */

void nodeCoordResetButtonCB (void)
{
  Widget button;
  XmString *stringTable;
  Arg args [3];
  int i;

  stringTable = createNodeCoordStringTable ();
```

```
    XtSetArg (args [0], XmNitems, stringTable);
    XtSetArg (args [1], XmNitemCount, NodeCount);
    XtSetArg (args [2], XmNselectedItemCount, 0);
    XtSetValues (nodeCoordList, args, 3);

    for (i = 0; i < NodeCount; i++)
      XmStringFree (stringTable[i]);
    XtFree ((char *) stringTable);

    clearXYZtextFields ();

    setEditButtonState ();

    /* Reset the environment & dimension */
    button = XtNameToWidget (nodeCoordShell, "nodeCoordForm.frame1.Environment");
    XtSetArg (args [0], XmNmenuHistory, nodeCoordEnvironment[Envindex]);
    XtSetValues (button, args, 1);
    button = XtNameToWidget (nodeCoordShell, "nodeCoordForm.frame2.Dimension");
    XtSetArg (args [0], XmNmenuHistory, nodeCoordDimension[DimIndex]);
    XtSetValues (button, args, 1);

  } /* end nodeCoordResetButtonCB */

  /*************************************************************************/
  static void clearXYZtextFields ()
  {
    Widget  x, y, z;

    getXYZtextFields (&x, &y, &z);
    XmTextSetString (x, "");
    XmTextSetString (y, "");
    XmTextSetString (z, "");

  } /* end clearXYZtextFields */

  /*************************************************************************/
  void nodeCoordEnvCB (w)
      Widget w;
  {
    Arg args [3];

    if (strcmp (XtName (w), "Ground Plane") == 0) {
      int n, i, count;
      XmString *items;
      char *string;

      n = 0;
      XtSetArg (args[n], XmNitems, &items); n++;
      XtSetArg (args[n], XmNitemCount, &count); n++;
      XtGetValues (nodeCoordList, args, n);

      for (i = 0; i < count; i++) {
        int node;
        float x, y, z;

        XmStringGetLtoR (items[i], XmSTRING_DEFAULT_CHARSET, &string);
        sscanf (string, "%d %f %f %f", &node, &x, &y, &z);
        XtFree ((char *) string);
        if (z < 0) {
          Widget menu;

          createMessageDialog (nodeCoordShell, "Node Coordinates",
                    "Z values cannot be less than zero\n\
  when Environment is a Ground Plane.");

          menu = XtNameToWidget (nodeCoordShell,
                    "nodeCoordForm.frame1.Environment");
          XtSetArg (args [0], XmNmenuHistory, nodeCoordEnvironment[FREE_SPACE]);
          XtSetValues (menu, args, 1);

          break;
        }
      }
    }
  } /* end nodeCoordEnvCB */

  /*************************************************************************/
  Boolean sufficientNodeCount(count)
  int count;
  {
    int i;
    extern int SWireCount, TaperWireCount, CWireCount,
          SurfacePatchCount, MultiplePatchCount;
    extern int *GW_END1, *GW_END2, *GC_END1, *GC_END2,
          *CW_END1, *CW_END2, *SP_Corner1, *SP_Corner2,
          *SP_Corner3, *SP_Corner4, *SM_Corner1,
          *SM_Corner2, *SM_Corner3;

    for (i = 0; i < SWireCount; i++)
      if (GW_END1[i] > count || GW_END2[i] > count)
        return False;
    for (i = 0; i < TaperWireCount; i++)
      if (GC_END1[i] > count || GC_END2[i] > count)
```

```
    return False;
  for (i = 0; i < CWireCount; i++)
    if (CW_END1[i] > count || CW_END2[i] > count)
      return False;
  for (i = 0; i < SurfacePatchCount; i++)
    if (SP_Corner1[i] > count || SP_Corner2[i] > count ||
        SP_Corner3[i] > count || SP_Corner4[i] > count)
      return False;
  for (i = 0; i < MultiplePatchCount; i++)
    if (SM_Corner1[i] > count || SM_Corner2[i] > count ||
        SM_Corner3[i] > count)
      return False;

  return True;
} /* end sufficientNodeCount */
```

fNodeCoord.c:

```
/*
 * Procedures for creating the Node Coordinates Window
 */

#include "control.h"
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/SelectioB.h>
#include <Xm/Text.h>
#include <stdio.h>

Widget      nodeCoordShell = NULL,
           *nodeCoordEnvironment,
           *nodeCoordDimension,
            nodeCoordList;

extern Widget  topLevel;
extern int     EnvIndex, DimIndex;
extern XmString *createNodeCoordStringTable ();
extern void    createOptionMenu ();
extern int     NodeCount;

/* Forward declarations for callbacks */
extern void    cancelButtonCB ();
extern void    HighlightText ();
extern void    nodeCoordAddButtonCB ();
extern void    nodeCoordApplyButtonCB ();
extern void    nodeCoordDeleteButtonCB ();
extern void    nodeCoordEnvCB ();
extern void    nodeCoordListCB ();
extern void    nodeCoordModifyButtonCB ();
extern void    nodeCoordOkButtonCB ();
extern void    nodeCoordResetButtonCB ();
extern void    nodeCoordTextCB ();

static Widget  createSelectionBox ();
static Widget  createWorkArea ();


/*****************************************************************************/
void createNodeCoordWindow ()

{
  Widget    form, frame, menuPane, button, optionMenu,
            selectionBox, workArea;
  Arg       args [10];
  int       n = 0;
  XmString  string;
  Position  x, y;
  extern void nodeCoordEnvCB ();
  extern void newEscapeAction();

  XtTranslateCoords (topLevel, (Position) 0, (Position) 0, &x, &y);
  XtSetArg (args [n], XmNx, x); n++;
  XtSetArg (args [n], XmNy, y + 100); n++;
  nodeCoordShell = XtCreatePopupShell ("Node Coordinates",
     topLevelShellWidgetClass, topLevel, args, 0);

  newEscapeAction(nodeCoordShell);

  form = XmCreateForm (nodeCoordShell, "nodeCoordForm", args, 0);

/* Create arrays to hold Option Widgets */

  nodeCoordEnvironment = (Widget *) XtMalloc (sizeof (Widget) * NUM_ENV);
  nodeCoordDimension = (Widget *) XtMalloc (sizeof (Widget) * NUM_DIM);

/* Create Environment Option Menu */

  n = 0;
```

```
    XtSetArg (args [n], XmNshadowType, XmSHADOW_ETCHED_IN);  n++;
    XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNleftOffset, 15); n++;
    XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNtopOffset, 15); n++;
    frame = XmCreateFrame (form, "frame1", args, n);
    XtManageChild (frame);

    menuPane = XmCreatePulldownMenu (frame, "menuPane", NULL, 0);

    button = XmCreatePushButton (menuPane, "Free Space", NULL, 0);
    XtManageChild (button);
    XtAddCallback (button, XmNactivateCallback, nodeCoordEnvCB, FREE_SPACE);
    nodeCoordEnvironment [0] = button;

    button = XmCreatePushButton (menuPane, "Ground Plane", NULL, 0);
    XtManageChild (button);
    XtAddCallback (button, XmNactivateCallback, nodeCoordEnvCB,
                   (XtPointer) GROUND_PLANE);
    nodeCoordEnvironment [1] = button;

    string = XmStringCreateSimple ("Environment");
    n = 0;
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetArg (args [n], XmNsubMenuId, menuPane); n++;
    optionMenu = XmCreateOptionMenu (frame, "Environment", args, n);
    XtManageChild (optionMenu);
    XtFree ((char *)string);

  /* Create Dimension Option Menu */

    n = 0;
    XtSetArg (args [n], XmNshadowType, XmSHADOW_ETCHED_IN);  n++;
    XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNrightOffset, 15); n++;
    XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNtopOffset, 15); n++;
    frame = XmCreateFrame (form, "frame2", args, n);
    XtManageChild (frame);

    menuPane = XmCreatePulldownMenu (frame, "menuPane", NULL, 0);

    button = XmCreatePushButton (menuPane, "Meters", NULL, 0);
    XtManageChild (button);
    nodeCoordDimension [0] = button;

    button = XmCreatePushButton (menuPane, "Centimeters", NULL, 0);
    XtManageChild (button);
    nodeCoordDimension [1] = button;

    button = XmCreatePushButton (menuPane, "Feet", NULL, 0);
    XtManageChild (button);
    nodeCoordDimension [2] = button;

    button = XmCreatePushButton (menuPane, "Inches", NULL, 0);
    XtManageChild (button);
    nodeCoordDimension [3] = button;

    string = XmStringCreateSimple ("Dimension");
    n = 0;
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetArg (args [n], XmNsubMenuId, menuPane); n++;
    optionMenu = XmCreateOptionMenu (frame, "Dimension", args, n);
    XtManageChild (optionMenu);
    XtFree ((char *)string);

  /* Create selection box & work area */

    selectionBox = createSelectionBox (form, frame);
    workArea = createWorkArea (selectionBox);
    XtManageChild (workArea);
    XtManageChild (selectionBox);
    XtManageChild (form);

  /* Select current item in selection box */

    if (NodeCount)
      XmListSelectPos (nodeCoordList, 1, TRUE);

  } /* end createNodeCoordWindow */

  /*************************************************************************/
  static Widget createSelectionBox (parent, widget)

    Widget   parent, widget;
  {
    Widget   box, child [2];
    Arg      args [15];
    int      n;
    XmString  string1, string2, string3;
    char     str [80];
    extern void newSelectActionTable ();
```

```
/* Create toplevel selection box */

  n = 0;
  sprintf (str, "%-20s%12s%12s%12s%s", "Node", "X", "Y", "Z", "  ");
  XtSetArg (args [n], XmNshadowThickness, 1); n++;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNleftOffset, 15); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNrightOffset, 15); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNtopWidget, widget); n++;
  XtSetArg (args [n], XmNtopOffset, 15); n++;
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNbottomOffset, 15); n++;
  XtSetArg (args [n], XmNlistVisibleItemCount, 5); n++;
  string1 = XmStringCreateSimple (str);
  XtSetArg (args [n], XmNlistLabelString, string1); n++;
  string2 = XmStringCreateSimple ("Cancel");
  XtSetArg (args [n], XmNhelpLabelString, string2); n++;
  string3 = XmStringCreateSimple ("Reset");
  XtSetArg (args [n], XmNcancelLabelString, string3); n++;
  XtSetArg (args [n], XmNlistVisibleItemCount, 5); n++;
  box = XmCreateSelectionBox (parent, "nodeCoordBox", args, n);

  XmStringFree (string1);
  XmStringFree (string2);
  XmStringFree (string3);

/* Register callbacks for SelectionBox list.  */

  nodeCoordList = XmSelectionBoxGetChild (box, XmDIALOG_LIST);

  n=0;
  XtSetArg(args[n], XmNselectionPolicy, XmEXTENDED_SELECT); n++;
  XtSetValues(nodeCoordList, args, n);
  XtAddCallback (nodeCoordList, XmNextendedSelectionCallback,
     nodeCoordListCB, NULL);
  newSelectActionTable (nodeCoordList);

/* Unmanage unneeded children */

  n = 0;
  child [n++] = XmSelectionBoxGetChild (box, XmDIALOG_SELECTION_LABEL);
  child [n++] = XmSelectionBoxGetChild (box, XmDIALOG_TEXT);
  XtUnmanageChildren (child, n);

  child [0] = XmSelectionBoxGetChild (box, XmDIALOG_APPLY_BUTTON);
  XtAddCallback (child [0], XmNactivateCallback,
     nodeCoordApplyButtonCB, NULL);
  XtManageChild (child [0]);

/* Add callbacks for ok, apply, reset, & cancel buttons */

  child [0] = XmSelectionBoxGetChild (box, XmDIALOG_OK_BUTTON);
  XtAddCallback (child [0], XmNactivateCallback, nodeCoordOkButtonCB, NULL);
  child [0] = XmSelectionBoxGetChild (box, XmDIALOG_CANCEL_BUTTON);
  XtAddCallback (child [0], XmNactivateCallback,
     nodeCoordResetButtonCB, NULL);
  child [0] = XmSelectionBoxGetChild (box, XmDIALOG_HELP_BUTTON);
  XtAddCallback (child [0], XmNactivateCallback, cancelButtonCB, NULL);

/* Remove default button */

  n = 0;
  XtSetArg (args [n], XmNdefaultButton, NULL); n++;
  XtSetValues (box, args, n);

  return (box);

} /* end createSelectionBox */

/*****************************************************************************/
static Widget createWorkArea (parent)

  Widget       parent;
{
  Widget       box,
               label,
               xText,
               yText,
               zText,
               button,
               mButton;
  Arg          args [10];
  int          n;
  XmString     string;

/* Create outer form box */

  n = 0;
  box = XmCreateForm (parent, "workArea", args, n);
  XtManageChild (box);
```

```
/* Create location-x text field */

  n = 0;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNleftOffset, 10); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNtopOffset, 30); n++;
  XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
  XtSetArg (args [n], XmNcolumns, 11); n++;
  XtSetArg (args [n], XmNmaxLength, 11); n++;
  xText = XmCreateText (box, "xText", args, n);
  XtManageChild (xText);

/* Create location-x label */

  n = 0;
  string = XmStringCreateSimple ("Location X:");
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNleftOffset, 10); n++;
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNbottomWidget, xText); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNrightWidget, xText); n++;
  label = XmCreateLabel (box, "xLabel", args, n);
  XtManageChild (label);
  XmStringFree (string);

/* Create location-y text field */

  n = 0;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNleftWidget, xText); n++;
  XtSetArg (args [n], XmNleftOffset, 5); n++;
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNbottomWidget, xText); n++;
  XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
  XtSetArg (args [n], XmNcolumns, 11); n++;
  XtSetArg (args [n], XmNmaxLength, 11); n++;
  yText = XmCreateText (box, "yText", args, n);
  XtManageChild (yText);

  XtAddCallback (xText, XmNactivateCallback, nodeCoordTextCB, yText);
  XtAddCallback (xText, XmNfocusCallback, HighlightText, NULL);

/* Create location-y label */

  n = 0;
  string = XmStringCreateSimple ("Y:");
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNleftWidget, yText); n++;
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNbottomWidget, label); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNrightWidget, yText); n++;
  label = XmCreateLabel (box, "yLabel", args, n);
  XtManageChild (label);
  XmStringFree (string);

/* Create location-z text field */

  n = 0;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNleftWidget, yText); n++;
  XtSetArg (args [n], XmNleftOffset, 5); n++;
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNbottomWidget, yText); n++;
  XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
  XtSetArg (args [n], XmNcolumns, 11); n++;
  XtSetArg (args [n], XmNmaxLength, 11); n++;
  zText = XmCreateText (box, "zText", args, n);
  XtManageChild (zText);

  XtAddCallback (yText, XmNactivateCallback, nodeCoordTextCB, zText);
  XtAddCallback (yText, XmNfocusCallback, HighlightText, NULL);

/* Create location-z label */

  n = 0;
  string = XmStringCreateSimple ("Z:");
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNleftWidget, zText); n++;
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNbottomWidget, label); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNrightWidget, zText); n++;
  label = XmCreateLabel (box, "zLabel", args, n);
  XtManageChild (label);
  XmStringFree (string);

/* Create Add button.  Put in dummy string of 8 characters so that
```

```
 * this button will be the same size as the others.  Then reset
 * label string.
 */
  n = 0;
  string = XmStringCreateSimple ("xxxxxx");
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNrightOffset, 10); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
  button = XmCreatePushButton (box, "addButton", args ,n);
  XtManageChild (button);
  XmStringFree (string);

  XtAddCallback (zText, XmNactivateCallback, nodeCoordTextCB, button);
  XtAddCallback (zText, XmNfocusCallback, HighlightText, NULL);

  string = XmStringCreateSimple ("Add");
  XtSetArg (args [n], XmNrecomputeSize, FALSE); n++;
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetValues (button, args, n);
  XmStringFree (string);

/* Register callback for add button.  */

  XtAddCallback (button, XmNactivateCallback, nodeCoordAddButtonCB, NULL);

/* Create Modify button */

  n = 0;
  string = XmStringCreateSimple ("Modify");
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNrightOffset, 10); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNtopWidget, button); n++;
  XtSetArg (args [n], XmNtopOffset, 10); n++;
  mButton = XmCreatePushButton (box, "modifyButton", args ,n);
  XtManageChild (mButton);
  XmStringFree (string);

  XtAddCallback (mButton, XmNactivateCallback, nodeCoordModifyButtonCB, NULL);

/* Create Delete button */

  n = 0;
  string = XmStringCreateSimple ("Delete");
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNrightOffset, 10); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNtopWidget, mButton); n++;
  XtSetArg (args [n], XmNtopOffset, 10); n++;
  button = XmCreatePushButton (box, "deleteButton", args ,n);
  XtManageChild (button);
  XmStringFree (string);

  XtAddCallback (button, XmNactivateCallback, nodeCoordDeleteButtonCB, NULL);

  return (box);

} /* end createWorkArea */
```

A.7 spiral.c

spiral.c:

```
/* routine spiral performs spiraling of the wires according
   to the option selected by the user
 */
#include <math.h>
#include <stdio.h>
#include "control.h"

void spiral(w, soption)
Widget w;
char *soption;
/* arguments need to be determined */
{
 int option, i, n, *array_sort, *ivector(), *newEnd1, *newEnd2,
    *newTag, *newNS;
 float *array_linear, *array_angle, *array_radial, *vector();
 long *array_total, *lvector();
 void longindsort(), XtFree_vector(), XtFree_ivector(), XtFree_lvector();
 float xc, yc, zc, lin_min, lin_max, ang_min, ang_max, rad_min, rad_max,
      val1, val2, val3, angle, lin_dif, ang_dif, rad_dif, *newRad;
 extern int SWireCount;
 extern float *X, *Y, *Z, *GW_RAD;
 extern int *GW_END1, *GW_END2, *GW_ITG, *GW_NS;
 extern void updateStraightWireWindow ();

 if (SWireCount < 2) return;

 n = SWireCount;
 option = atoi (soption);
/* allocate storage for the vector arrays */
 array_sort = ivector(1,n);
 array_linear = vector(1,n);
 array_angle = vector(1,n);
 array_radial = vector(1,n);
 array_total = lvector(1,n);
/* loop through the wires */
 for (i = 1; i < n+1; i++)
 {
/* determine the center point of the wire, put it in (xc,yc,zc) */
   xc = (X[GW_END1[i-1]-1] + X[GW_END2[i-1]-1]) / 2;
   yc = (Y[GW_END1[i-1]-1] + Y[GW_END2[i-1]-1]) / 2;
   zc = (Z[GW_END1[i-1]-1] + Z[GW_END2[i-1]-1]) / 2;
/* depending on the parameter "option", assign values to the arrays */
   switch (option)
   {
    case 1:
    case 2:  array_linear[i] = xc;
          angle = (zc == 0) && (yc == 0) ? 0.0 : atan2(zc,yc);
          if (angle < 0.0) angle = angle + 2.0*M_PI;
          array_angle[i] = angle;
          array_radial[i] = sqrt(zc*zc+yc*yc);
          break;
    case 3:
    case 4:  array_linear[i] = yc;
          angle = (zc == 0) && (xc == 0) ? 0.0 : atan2(zc,xc);
          if (angle < 0.0) angle = angle + 2.0*M_PI;
          array_angle[i] = angle;
          array_radial[i] = sqrt(zc*zc+xc*xc);
          break;
    case 5:
    case 6:  array_linear[i] = zc;
          angle = (xc == 0) && (yc == 0) ? 0.0 : atan2(yc,xc);
          if (angle < 0.0) angle = angle + 2.0*M_PI;
          array_angle[i] = angle;
          array_radial[i] = sqrt(xc*xc+yc*yc);
   }
/* initialize the min and max values */
   if (i == 1)
   {
    lin_min = array_linear[1];
    lin_max = array_linear[1];
    ang_min = array_angle[1];
    ang_max = array_angle[1];
    rad_min = array_radial[1];
    rad_max = array_radial[1];
   }
   else
   {
/* set min and max values depending on the parameter "option" */
    switch (option)
    {
     case 1:
     case 3:
     case 5:if (array_linear[i] < lin_min) lin_min = array_linear[i];
          if (array_linear[i] > lin_max) lin_max = array_linear[i];
          if (array_angle[i] < ang_min) ang_min = array_angle[i];
          if (array_angle[i] > ang_max) ang_max = array_angle[i];
          if (array_radial[i] < rad_min) rad_min = array_radial[i];
```

```
            if (array_radial[i] > rad_max) rad_max = array_radial[i];
            break;
        case 2:
        case 4:
        case 6:if (array_linear[i] > lin_min) lin_min = array_linear[i];
            if (array_linear[i] < lin_max) lin_max = array_linear[i];
            if (array_angle[i] > ang_min) ang_min = array_angle[i];
            if (array_angle[i] < ang_max) ang_max = array_angle[i];
            if (array_radial[i] < rad_min) rad_min = array_radial[i];
            if (array_radial[i] > rad_max) rad_max = array_radial[i];
      }
     }
    }
    rad_dif = 1000.0/(rad_max - rad_min);
    ang_dif = 999000.0/(ang_max - ang_min);
    lin_dif = 999000000.0/(lin_max - lin_min);
  /* now determine array_total */
  /* loop through the wires */
    for (i = 1; i < n+1; i++) {
     val1 = rad_max == rad_min ? 0.0 : rad_dif * (array_radial[i] - rad_min);
     val2 = ang_max == ang_min ? 1000.0 :
      1000.0 + ang_dif * (array_angle[i] - ang_min);
     val3 = lin_max == lin_min ? 1000000.0 :
      1000000.0 + lin_dif * (array_linear[i] - lin_min);
     array_total[i] = (long) (val1 + val2 + val3);
    }
  /* now do an indexed sort on array_total */
    longindsort(n,array_total,array_sort);

  /* Free some memory */
    XtFree_vector(array_linear,1);
    XtFree_vector(array_angle,1);
    XtFree_vector(array_radial,1);
    XtFree_lvector(array_total,1);

  /* now use array_sort as an index array to sort the wires */
    newEnd1 = (int *) XtMalloc (sizeof (int) * SWireCount);
    newEnd2 = (int *) XtMalloc (sizeof (int) * SWireCount);
    newNS = (int *) XtMalloc (sizeof (int) * SWireCount);
    newTag = (int *) XtMalloc (sizeof (int) * SWireCount);
    newRad = (float *) XtMalloc (sizeof (float) * SWireCount);
    for (i = 0; i < SWireCount; i++) {
     int j;
     j = array_sort[i + 1] - 1;
     newEnd1[i] = GW_END1[j];
     newEnd2[i] = GW_END2[j];
     newNS[i] = GW_NS[j];
     newTag[i] = GW_ITG[j];
     newRad[i] = GW_RAD[j];
    }
    XtFree ((char *) GW_END1);
    XtFree ((char *) GW_END2);
    XtFree ((char *) GW_NS);
    XtFree ((char *) GW_ITG);
    XtFree ((char *) GW_RAD);
    GW_END1 = newEnd1;
    GW_END2 = newEnd2;
    GW_NS = newNS;
    GW_ITG = newTag;
    GW_RAD = newRad;

  /* Update the Straight Wires window if open */
    updateStraightWireWindow ();

  /* Free array_sort */
    XtFree_ivector(array_sort,1);
  /* Make compiler happy */
    w = w;
  }

  void longindsort(n,arrin,indx)
  int n,indx[];
  long arrin[];
  {
    int l,j,ir,indxt,i;
        long q;

    for (j=1;j<=n;j++) indx[j]=j;
    if (n == 1) return;
    l=(n >> 1) + 1;
    ir=n;
    for (;;) {
     if (l > 1)
      q=arrin[(indxt=indx[--l])];
     else {
      q=arrin[(indxt=indx[ir])];
      indx[ir]=indx[1];
      if (--ir == 1) {
       indx[1]=indxt;
       return;
      }
     }
     i=l;
```

```
    j=l << 1;
    while (j <= ir) {
     if (j < ir && arrin[indx[j]] < arrin[indx[j+1]]) j++;
     if (q < arrin[indx[j]]) {
      indx[i]=indx[j];
      j += (i=j);
     }
     else j=ir+1;
    }
    indx[i]=indxt;
  }
}

void nrerror(error_text)
char error_text[];
/* standard error handler */
{
  void exit();

  fprintf(stderr,"Run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
}

float *vector(nl,nh)
int nl,nh;
/* allocates a float vector with range [nl..nh] */
{
  float *v;

  v=(float *)XtMalloc((unsigned) (nh-nl+1)*sizeof(float));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
/* allocates an int vector with range [nl..nh] */
{
  int *v;

  v=(int *)XtMalloc((unsigned) (nh-nl+1)*sizeof(int));
  if (!v) nrerror("allocation failure in ivector()");
  return v-nl;
}

long *lvector(nl,nh)
int nl,nh;
/* allocates an long vector with range [nl..nh] */
{
  long *v;

  v=(long *)XtMalloc((unsigned) (nh-nl+1)*sizeof(long));
  if (!v) nrerror("allocation failure in lvector()");
  return v-nl;
}

void XtFree_vector(v,nl)
float *v;
int nl;
/* XtFrees a float vector allocated by vector(). */
{
  XtFree((char*) (v+nl));
}

void XtFree_ivector(v,nl)
int *v,nl;
/* XtFrees a int vector allocated by ivector(). */
{
  XtFree((char*) (v+nl));
}

void XtFree_lvector(v,nl)
long *v;
int nl;
/* XtFrees a long vector allocated by lvector(). */
{
  XtFree((char*) (v+nl));
}
```

A.8 cMeshes.c, fMeshes.c

cMeshes.c:

```
/*
 * Filename: cMeshes.c
 * Callbacks & procedures for the Meshes form
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <X11/IntrinsicP.h>
#include <X11/ShellP.h>
#include <Xm/Xm.h>
#include <Xm/List.h>
#include <Xm/MessageB.h>
#include <Xm/SelectioB.h>
#include <Xm/Text.h>
#include "control.h"

#define LENGTH(x,y,z) (sqrt((x)*(x)+(y)*(y)+(z)*(z)))

extern Widget meshesShell, meshesRowColumn, meshesIndexLabel;

extern float *X, *Y, *Z;
extern int *GW_END1, *GW_END2, *GW_NS, *GW_ITG;
extern float *GW_RAD;
extern int *GN_IPERF;

/* variables for holding meshes data */
static int *corner1, *corner2, *corner3, *corner4, *number12, *number23;
int MeshesCount = 0;
static float *factor;
static int oldSWireCount;
static int numNodes, numWires;
static Boolean okFlag;  /* If set, then ok button was pressed */

extern void createMeshesWindow ();
extern int getListPosition ();
extern int getListCount ();
extern void modifyMeshesListAll ();

static void clearMeshesTextFields ();
static void continueCB ();
static void createQuestionDialog ();
static void createWiresFromInputs ();
static Boolean createXmStringFromInputs ();
static void getMeshesTextFields ();
static Boolean plane ();
static void removeRedundantItems ();
static void setEditButtonState ();
static void stopCB ();

/*****************************************************************************/
static Boolean plane ()
{
 int i, corners [4];
 float diff, dot, x, angle32, angle43, angle42,
     xc [4], yc [4], zc [4], dl [4];
 Widget c1text, c2text, c3text, c4text, dummy;
 char *c1, *c2, *c3, *c4;

 /* Get text field widgets */
 getMeshesTextFields (&c1text, &c2text, &c3text, &c4text, &dummy,
             &dummy, &dummy);

 /* Get the values */
 corners[0] = atoi (c1 = XmTextGetString (c1text));
 corners[1] = atoi (c2 = XmTextGetString (c2text));
 corners[2] = atoi (c3 = XmTextGetString (c3text));
 corners[3] = atoi (c4 = XmTextGetString (c4text));
 XtFree (c1);
 XtFree (c2);
 XtFree (c3);
 XtFree (c4);

 /* Check for corners with the same node */
 if (corners[0] == corners[1]) {
  createMessageDialog (meshesShell, "Error",
             "Corners #1 and #2 cannot be equivalent.",
             XmDIALOG_ERROR);
  return (1);
 } else if (corners[1] == corners[2]) {
  createMessageDialog (meshesShell, "Error",
             "Corners #2 and #3 cannot be equivalent.",
             XmDIALOG_ERROR);
  return (1);
 } else if (corners[0] == corners[2]) {
  createMessageDialog (meshesShell, "Error",
             "Corners #1 and #3 cannot be equivalent.",
```

```
                    XmDIALOG_ERROR);
      return (1);
    } else if (corners[0] == corners[3]) {
      createMessageDialog (meshesShell, "Error",
                    "Corners #1 and #4 cannot be equivalent.",
                    XmDIALOG_ERROR);
      return (1);
    } else if (corners[1] == corners[3]) {
      createMessageDialog (meshesShell, "Error",
                    "Corners #2 and #4 cannot be equivalent.",
                    XmDIALOG_ERROR);
      return (1);
    }

    /* Check to see if surface is flat (only if not a triangle) */
    if (corners[2] != corners[3]) {
      for (i = 0; i < 4; i++) {
        int node = corners[i] - 1;
        if (i > 0) {
          xc[i] = X[node] - xc[0];
          yc[i] = Y[node] - yc[0];
          zc[i] = Z[node] - zc[0];
          dl[i] = LENGTH (xc[i], yc[i], zc[i]);
        } else {
          xc[0] = X[node];
          yc[0] = Y[node];
          zc[0] = Z[node];
        }
      }

      /* Find angle between corners 3 and 2 */
      dot = xc[2] * xc[1] + yc[2] * yc[1] + zc[2] * zc[1];
      x = dot / (dl[2] * dl[1]);
      angle32 = atan (-x / sqrt (1 - x*x)) + M_PI / 2;

      /* Find angle between corners 4 and 3 */
      dot = xc[3] * xc[2] + yc[3] * yc[2] + zc[3] * zc[2];
      x = dot / (dl[3] * dl[2]);
      angle43 = atan (-x / sqrt (1 - x*x)) + M_PI / 2;

      /* Find angle between corners 4 and 2 */
      dot = xc[3] * xc[1] + yc[3] * yc[1] + zc[3] * zc[1];
      if (dot == 0)
        angle42 = M_PI / 2;
      else {
        x = dot / (dl[3] * dl[1]);
        angle42 = atan (-x / sqrt (1 - x*x)) + M_PI / 2;
      }
      diff = abs (angle42 - (angle43 + angle32));
      if (diff > .00001745) { /* with .001 degrees assume flat */
        createMessageDialog (meshesShell, "Error",
                    "Region is not planar.",
                    XmDIALOG_ERROR);
        return (1);
      }
    }
    return (0);

} /* end plane */

/*****************************************************************************/
void openMeshesWindow ()

{
  Widget box, list;
  ShellWidget sw = (ShellWidget) meshesShell;

  if (meshesShell == NULL)
    createMeshesWindow ();
  else if (sw->shell.popped_up)
    return;  /* Don't do anything if already open */

  XtPopup (meshesShell, XtGrabNone);

  box = XtNameToWidget (meshesShell, "meshesForm.meshesBox");
  list = XmSelectionBoxGetChild (box, XmDIALOG_LIST);
  setEditButtonState (list);

} /* end openMeshesWindow */

/*****************************************************************************/
void meshesListCB (w, clientData, cb)
  Widget w;
  XtPointer clientData;
  XmListCallbackStruct *cb;
{
  char corner1 [15], corner2 [15], corner3 [15], corner4 [15],
       number12 [15], number23 [15], factor [15];
  char *string;
  Widget corner1Widget, corner2Widget, corner3Widget, corner4Widget,
         number12Widget, number23Widget, factorWidget;

  if (cb->selected_item_count == 1) {
```

```
    XmStringGetLtoR (cb->selected_items[0], XmSTRING_DEFAULT_CHARSET, &string);

   /* Get text field widgets */
    getMeshesTextFields (&corner1Widget, &corner2Widget, &corner3Widget,
                &corner4Widget, &number12Widget, &number23Widget,
                &factorWidget);

    /* Get data & put into text fields */
    sscanf (string, "%s %s %s %s %s %s %s", corner1, corner2, corner3,
          corner4, number12, number23, factor);
    XmTextSetString (corner1Widget, corner1);
    XmTextSetString (corner2Widget, corner2);
    XmTextSetString (corner3Widget, corner3);
    XmTextSetString (corner4Widget, corner4);
    XmTextSetString (number12Widget, number12);
    XmTextSetString (number23Widget, number23);
    XmTextSetString (factorWidget, factor);
    XtFree (string);
   }

   /* Enable edit buttons */
   setEditButtonState (w);

   clientData = clientData; /* Make compiler happy */

} /* end meshesListCB */

/*****************************************************************************/
static void setEditButtonState (listWidget)

  Widget listWidget;
{
  Widget modifyButton, deleteButton, corner1Text, corner2Text,
       corner3Text, corner4Text, number12Text, number23Text,
       factorText, addButton;
  int index, count;

  /* Get the modify & delete buttons. */
  modifyButton = XtNameToWidget (meshesShell,
                "meshesForm.meshesBox.workArea.modifyButton");
  deleteButton = XtNameToWidget (meshesShell,
                "meshesForm.meshesBox.workArea.deleteButton");
  addButton = XtNameToWidget (meshesShell,
                "meshesForm.meshesBox.workArea.addButton");

  count = getListCount (listWidget);
  if (count == 1) {
    XtSetSensitive (addButton, TRUE);
    XtSetSensitive (modifyButton, TRUE);
    XtSetSensitive (deleteButton, TRUE);
    index = getListPosition (listWidget);
  } else {
    index = 0;
    getMeshesTextFields (&corner1Text, &corner2Text,
       &corner3Text, &corner4Text, &number12Text, &number23Text,
       &factorText);
    XmTextSetString (corner1Text, "");
    XmTextSetString (corner2Text, "");
    XmTextSetString (corner3Text, "");
    XmTextSetString (corner4Text, "");
    XmTextSetString (number12Text, "1");
    XmTextSetString (number23Text, "1");
    XmTextSetString (factorText, "");
    if (count > 1) {
      XtSetSensitive (addButton, FALSE);
      XtSetSensitive (modifyButton, TRUE);
      XtSetSensitive (deleteButton, TRUE);
    }
    else {
      XtSetSensitive (addButton, TRUE);
      XtSetSensitive (modifyButton, FALSE);
      XtSetSensitive (deleteButton, FALSE);
    }
  }

  XtVaGetValues (listWidget, XmNitemCount, &count, NULL);
  updateIndexLabel (meshesIndexLabel, index, count);

} /* end setEditButtonState */

/*****************************************************************************/
static void getMeshesTextFields (corner1Text, corner2Text,
    corner3Text, corner4Text, number12Text, number23Text, areaFactorText)

  Widget *corner1Text, *corner2Text, *corner3Text, *corner4Text,
       *number12Text, *number23Text, *areaFactorText;
{
  /* Get the Meshes  text fields */
  *corner1Text = XtNameToWidget (meshesRowColumn,"corner1Text");
  *corner2Text = XtNameToWidget (meshesRowColumn,"corner2Text");
  *corner3Text = XtNameToWidget (meshesRowColumn,"corner3Text");
  *corner4Text = XtNameToWidget (meshesRowColumn,"corner4Text");
  *number12Text = XtNameToWidget (meshesRowColumn,"number12Text");
```

```
    *number23Text = XtNameToWidget (meshesRowColumn,"number23Text");
    *areaFactorText = XtNameToWidget (meshesRowColumn,"factorText");

  } /* end getMeshesTextFields */

  /*****************************************************************/
  static Boolean createXmStringFromInputs (xmString)
    XmString *xmString;
  {
    Widget corner1Text, corner2Text, corner3Text, corner4Text,
         number12Text, number23Text, areaFactorText;
    char string [85], *corner1, *corner2, *corner3, *corner4,
       *number12, *number23, *areaFactor;
    int num12, num23, corn1, corn2, corn3, corn4;
    float factor;
    extern Boolean valueCheck ();
    extern int NodeCount;

    getMeshesTextFields (&corner1Text, &corner2Text, &corner3Text,
                &corner4Text, &number12Text, &number23Text,
                &areaFactorText);

    /* Check for invalid inputs */
    corn1 = atoi (corner1 = XmTextGetString (corner1Text));
    corn2 = atoi (corner2 = XmTextGetString (corner2Text));
    corn3 = atoi (corner3 = XmTextGetString (corner3Text));
    corn4 = atoi (corner4 = XmTextGetString (corner4Text));
    num12 = atoi (number12 = XmTextGetString (number12Text));
    num23 = atoi (number23 = XmTextGetString (number23Text));
    factor = atof (areaFactor = XmTextGetString (areaFactorText));
    if (!valueCheck (meshesShell, "number of patches from 1 to 2",
          num12 > 0) ||
       !valueCheck (meshesShell, "number of patches from 2 to 3",
          num23 > 0) ||
       !valueCheck (meshesShell, "corner 1",
          (corn1 > 0) && (corn1 <= NodeCount)) ||
       !valueCheck (meshesShell, "corner 2",
          (corn2 > 0) && (corn2 <= NodeCount)) ||
       !valueCheck (meshesShell, "corner 3",
          (corn2 > 0) && (corn3 <= NodeCount)) ||
       !valueCheck (meshesShell, "corner 3",
          (corn4 > 0) && (corn4 <= NodeCount)))
      return (False);

    /* Create a string using inputs */
    sprintf (string, "%10d%10d%10d%10d%12d%12d%12g",
          corn1, corn2, corn3, corn4, num12, num23, factor);
    XtFree ((char *) corner1);
    XtFree ((char *) corner2);
    XtFree ((char *) corner3);
    XtFree ((char *) corner4);
    XtFree ((char *) number12);
    XtFree ((char *) number23);
    XtFree ((char *) areaFactor);
    *xmString = XmStringCreateSimple (string);
    return (True);

  } /* end createXmStringFromInputs */

  /*****************************************************************/
  void meshesAddButtonCB (w, listWidget)
    Widget w, listWidget;
  {
    int n, iWireIndex, count;
    XmString xmString, *items;
    Widget text;
    Arg args [2];

    /* Intrinsic diagnostics */
    if (plane ()) return;

    /* Create a string from the inputs then check for duplicates */
    if (createXmStringFromInputs (&xmString)) {

      /* Get items already in list */
      n = 0;
      XtSetArg (args [n], XmNitems, &items); n++;
      XtSetArg (args [n], XmNitemCount, &count); n++;
      XtGetValues (listWidget, args, n);

      for (n = 0; n < count; n++) {
        if (XmStringCompare (xmString, items [n])) {
          createMessageDialog (meshesShell, "Error",
                "Mesh description already in list.",
                XmDIALOG_ERROR);
          return;
        }
      }

      /* Get the current list selection */
      iWireIndex = getListPosition (listWidget) + 1;

      /* Insert string into list */
```

```
    XmListAddItem (listWidget, xmString, iWireIndex);
    XmListSelectPos (listWidget, iWireIndex, TRUE);
    XmStringFree (xmString);

    /* Move focus to corner1 text field */
    text = XtNameToWidget (meshesRowColumn,"corner1Text");
    XtSetKeyboardFocus (meshesShell, text);
  }

  /* Make compiler happy */
  w = w;

  XmListSetPos(listWidget, iWireIndex);

} /* end meshesAddButtonCB */

/*****************************************************************************/
void meshesModifyButtonCB (w, listWidget)
  Widget w, listWidget;
{
  int iWireIndex;
  XmString xmString;

  if (getListCount (listWidget) > 1) {
    modifyMeshesListAll (listWidget);
    setEditButtonState (listWidget);
    return;
  }

  /* Intrinsic diagnostics */
  if (plane ()) return;

  /* Create a string from the inputs then check for duplicates */
  if (createXmStringFromInputs (&xmString)) {
    int n, count;
    Arg args [2];
    XmString *items;

    /* Get items already in list */
    n = 0;
    XtSetArg (args [n], XmNitems, &items); n++;
    XtSetArg (args [n], XmNitemCount, &count); n++;
    XtGetValues (listWidget, args, n);

    for (n = 0; n < count; n++) {

      if (XmStringCompare (xmString, items [n])) {
        createMessageDialog (meshesShell, "Error",
                             "Mesh description already in list.",
                             XmDIALOG_ERROR);
        return;
      }
    }

    /* Get the current list selection */
    if (iWireIndex = getListPosition (listWidget)) {

      /* Replace string in list */
      XmListReplaceItemsPos (listWidget, &xmString, 1, iWireIndex);
      XmListSelectPos (listWidget, iWireIndex, TRUE);
      XmStringFree (xmString);
    }
  }

  w = w;  /* Make compiler happy */

} /* end meshesModifyButtonCB */

/*****************************************************************************/
void meshesDeleteButtonCB (w, listWidget)
  Widget w, listWidget;
{
  int iWireIndex;
  int count, *positionList;

  /* Clear the text fields */
  clearMeshesTextFields ();

  /* Get the current list selection */
  if (iWireIndex = getListPosition (listWidget)) {

    /* Delete string in list */
    XmListGetSelectedPos (listWidget, &positionList, &count);
    XmListDeletePositions (listWidget, positionList, count);
    XtFree ((char *)positionList);
    setEditButtonState (listWidget);
    XmListSelectPos (listWidget, iWireIndex, TRUE);
    if (!XmListPosSelected(listWidget, iWireIndex))
      XmListSelectPos (listWidget, iWireIndex - 1, TRUE);
  }

  w = w;  /* Make compiler happy */
```

```
} /* end meshesDeleteButtonCB */

/*****************************************************************/
void meshesTextCB (w, nextText)
    Widget w, nextText;
{
 XtSetKeyboardFocus (meshesShell, nextText);

 w = w;  /* Make compiler happy */

} /*end meshesTextCB */

/*****************************************************************
 * Save the changes and close the window.
 */

void meshesOkButtonCB (w, list)
    Widget w, list;
{
 extern void meshesApplyButtonCB ();

 meshesApplyButtonCB (w, list);
 okFlag = True;
 if (MeshesCount <= 0)
  XtPopdown (meshesShell);

} /* end meshesOkButtonCB */

/*****************************************************************
 * Save the changes and leave window open.
 */
void meshesApplyButtonCB (w, list)
 Widget w, list;
{
 int i;
 XmString *items;
 Arg args [2];
 char msg [80];
 extern int NodeCount, SWireCount;
 extern Boolean saveAlert;

 okFlag = False;

 /* Get list items & count */
 i = 0;
 XtSetArg (args [i], XmNitems, &items); i++;
 XtSetArg (args [i], XmNitemCount, &MeshesCount); i++;
 XtGetValues (list, args, i);

 if (MeshesCount <= 0) return;

 /* Allocate memory for input values */
 corner1 = (int *) XtMalloc (sizeof (int) * MeshesCount);
 corner2 = (int *) XtMalloc (sizeof (int) * MeshesCount);
 corner3 = (int *) XtMalloc (sizeof (int) * MeshesCount);
 corner4 = (int *) XtMalloc (sizeof (int) * MeshesCount);
 number12 = (int *) XtMalloc (sizeof (int) * MeshesCount);
 number23 = (int *) XtMalloc (sizeof (int) * MeshesCount);
 factor = (float *) XtMalloc (sizeof (float) * MeshesCount);

 /* Fill arrays with input values */
 for (i = 0; i < MeshesCount; i++) {
  char *string, c1string [15], c2string [15], c3string [15],
      c4string [15], n12string [15], n23string [15], fString [15];

  /* Get data & put into text fields */
  XmStringGetLtoR (items[i], XmSTRING_DEFAULT_CHARSET, &string);
  sscanf (string, "%s %s %s %s %s %s %s",
        c1string, c2string, c3string, c4string,
        n12string, n23string, fString);
  corner1 [i] = atoi (c1string);
  corner2 [i] = atoi (c2string);
  corner3 [i] = atoi (c3string);
  corner4 [i] = atoi (c4string);
  number12 [i] = atoi (n12string);
  number23 [i] = atoi (n23string);
  factor [i] = atof (fString);
  XtFree (string);
 }

 /* Save old value for SWireCount */
 oldSWireCount = SWireCount;

 /* Indicate to the user the number of nodes that will be added
  * (not used in NEC)
  * Determine approximation of maximum number of nodes and number of
  * wires that will be added
  */
 numWires = SWireCount;
 numNodes = NodeCount;
 for (i = 0; i < MeshesCount; i++) {
  numNodes += (number23[i] + 1) * (number12[i] + 1);
  numWires += 2 * number12[i] * number23[i] + number23[i] + number12[i];
```

```
    }

    /* Ask user if this number of wires should be added */
    sprintf (msg,
          "Approximately %d wires will result.\nDo you want to continue?",
          numWires - SWireCount);
    createQuestionDialog (msg, list);

    w = w;  /* Make compiler happy */

    saveAlert = True;

} /* end meshesApplyButtonCB */

/*****************************************************************
 * Clear all list items.
 */

void meshesResetButtonCB (w, list)
  Widget w, list;
{
  /* Clear the list & reset the list item count */
  XmListDeleteAllItems (list);
  MeshesCount = 0;

  /* Clear all text fields & reset the state of the edit buttons */
  clearMeshesTextFields ();
  setEditButtonState (list);

  w = w;  /* Make compiler happy */

} /* end meshesResetButtonCB */

/*****************************************************************/
static void clearMeshesTextFields ()

{
  Widget corner1, corner2, corner3, corner4,
       number12, number23, areaFactor;

  getMeshesTextFields (&corner1, &corner2, &corner3, &corner4,
              &number12, &number23, &areaFactor);
  XmTextSetString (corner1, "");
  XmTextSetString (corner2, "");
  XmTextSetString (corner3, "");
  XmTextSetString (corner4, "");
  XmTextSetString (number12, "");
  XmTextSetString (number23, "");
  XmTextSetString (areaFactor, "");

} /* end clearMeshesTextFields */

/*****************************************************************/
static void createQuestionDialog (message, list)
    char *message;
    Widget list;
{
  Widget dialog;
  XmString msg, yes, no, title;

  dialog = XmCreateQuestionDialog (meshesShell, "dialog", NULL, 0);
  yes = XmStringCreateSimple ("Yes");
  no = XmStringCreateSimple ("No");
  msg = XmStringCreateLtoR (message, XmSTRING_DEFAULT_CHARSET);
  title = XmStringCreateSimple ("Meshes Question");

  XtVaSetValues (dialog,
            XmNdialogTitle, title,
            XmNmessageString, msg,
            XmNokLabelString, yes,
            XmNcancelLabelString, no,
            XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
            NULL);

  XtUnmanageChild
    (XmMessageBoxGetChild (dialog, XmDIALOG_HELP_BUTTON));

  XtAddCallback (dialog, XmNokCallback, continueCB, (XtPointer) list);
  XtAddCallback (dialog, XmNcancelCallback, stopCB, (XtPointer) NULL);

  XmStringFree (yes);
  XmStringFree (no);
  XmStringFree (msg);
  XmStringFree (title);

  XtManageChild (dialog);
  XtPopup (XtParent (dialog), XtGrabNone);

} /* end createQuestionDialog */

/*****************************************************************/
static void continueCB (w, list)
    Widget w, list;
```

```
{
  extern void meshesResetButtonCB ();

  createWiresFromInputs ();
  meshesResetButtonCB (w, list);
  XtDestroyWidget (XtParent (w));
  if (okFlag)
    XtPopdown (meshesShell);

} /* end questionCB */

/*****************************************************************/
static void stopCB (w)
    Widget w;
{
  /* Free allocated memory */
  XtFree ((char *) corner1);
  XtFree ((char *) corner2);
  XtFree ((char *) corner3);
  XtFree ((char *) corner4);
  XtFree ((char *) number12);
  XtFree ((char *) number23);
  XtFree ((char *) factor);
  XtDestroyWidget (XtParent (w));

} /* end questionCB */

/*****************************************************************/
static void createWiresFromInputs ()
{
  int i, jn, jw, k, j, iv, ih;
  float radius, maxradius, s21x, s21y, s21z, s23x, s23y, s23z,
       s13x, s13y, s13z, l12, l23, l31, s1, s2, area1, area2, areaTotal,
       s34x, s34y, s34z, s41x, s41y, s41z, l41, l34, l13,
       xb, yb, zb, xt, yt, zt, dx, dy, dz, ltotal;
  extern int NodeCount, SWireCount;

  /* Allocate memory for new nodes & new wires */
  X = (float *) XtRealloc ((char *) X, sizeof (float) * numNodes);
  Y = (float *) XtRealloc ((char *) Y, sizeof (float) * numNodes);
  Z = (float *) XtRealloc ((char *) Z, sizeof (float) * numNodes);
  GW_END1 = (int *) XtRealloc ((char *) GW_END1, sizeof (int) * numWires);
  GW_END2 = (int *) XtRealloc ((char *) GW_END2, sizeof (int) * numWires);
  GW_NS = (int *) XtRealloc ((char *) GW_NS, sizeof (int) * numWires);
  GW_ITG = (int *) XtRealloc ((char *) GW_ITG, sizeof (int) * numWires);
  GW_RAD = (float *) XtRealloc ((char *) GW_RAD, sizeof (float) * numWires);

  maxradius = 0;

  /* Determine geometry nodes for mesh */
  for (i = 0; i < MeshesCount; i++) {
    jn = NodeCount;

    /* Find area of individual mesh region */
    s21x = X[corner2[i] - 1] - X[corner1[i] - 1];
    s21y = Y[corner2[i] - 1] - Y[corner1[i] - 1];
    s21z = Z[corner2[i] - 1] - Z[corner1[i] - 1];
    s23x = X[corner3[i] - 1] - X[corner2[i] - 1];
    s23y = Y[corner3[i] - 1] - Y[corner2[i] - 1];
    s23z = Z[corner3[i] - 1] - Z[corner2[i] - 1];
    s13x = X[corner3[i] - 1] - X[corner1[i] - 1];
    s13y = Y[corner3[i] - 1] - Y[corner1[i] - 1];
    s13z = Z[corner3[i] - 1] - Z[corner1[i] - 1];
    l12 = LENGTH (s21x, s21y, s21z);
    l23 = LENGTH (s23x, s23y, s23z);
    l31 = LENGTH (s13x, s13y, s13z);
    s1 = (l12 + l23 + l31) / 2;
    area1 = sqrt (s1 * (s1 - l12) * (s1 - l23) * (s1 - l31));
    s34x = X[corner3[i] - 1] - X[corner4[i] - 1];
    s34y = Y[corner3[i] - 1] - Y[corner4[i] - 1];
    s34z = Z[corner3[i] - 1] - Z[corner4[i] - 1];
    s41x = X[corner1[i] - 1] - X[corner4[i] - 1];
    s41y = Y[corner1[i] - 1] - Y[corner4[i] - 1];
    s41z = Z[corner1[i] - 1] - Z[corner4[i] - 1];
    l34 = LENGTH (s34x, s34y, s34z);
    l41 = LENGTH (s41x, s41y, s41z);
    l13 = l31;
    s2 = (l34 + l41 + l13) / 2;
    area2 = sqrt (s2 * (s2 - l34) * (s2 - l41) * (s2 - l13));
    areaTotal = area1 + area2;
    s21x = s21x / number12[i];
    s21y = s21y / number12[i];
    s21z = s21z / number12[i];
    s34x = s34x / number12[i];
    s34y = s34y / number12[i];
    s34z = s34z / number12[i];

    /* Determine description of geometry nodes.
     * dn[j][i] - dimensions of node (j = 1, 2, 3) i
     */
    for (ih = 0; ih <= number12[i]; ih++) {

      /* Position of bottom geometry nodes */
```

```
      xb = X[corner1[i] - 1] + ih * s21x;
      yb = Y[corner1[i] - 1] + ih * s21y;
      zb = Z[corner1[i] - 1] + ih * s21z;
      xt = X[corner4[i] - 1] + ih * s34x;
      yt = Y[corner4[i] - 1] + ih * s34y;
      zt = Z[corner4[i] - 1] + ih * s34z;
      dx = (xt - xb) / number23[i];
      dy = (yt - yb) / number23[i];
      dz = (zt - zb) / number23[i];
      for (iv = 0; iv <= number23[i]; iv++) {

        /* Geometry nodes up from bottom node */
        X[jn] = xb + iv * dx;
        Y[jn] = yb + iv * dy;
        Z[jn] = zb + iv * dz;
        jn++;
      }
    }

    /* Determine description of vertical & horizontal wires */
    k = NodeCount;
    jw = SWireCount;
    ltotal = 0;
    for (ih = 0; ih <= number12[i]; ih++) {
      for (iv = 1; iv <= number23[i]; iv++) {
        k++;
        for (j = 1; j <= 2; j++) {
          if (j == 1) {
            GW_END1[jw] = k;
            GW_END2[jw] = k + 1;
          } else if (j == 2 && ih < number12[i]) {
            GW_END1[jw] = k;
            GW_END2[jw] = k + number23[i] + 1;
          } else
            break;
          dx = X[GW_END2[jw] - 1] - X[GW_END1[jw] - 1];
          dy = Y[GW_END2[jw] - 1] - Y[GW_END1[jw] - 1];
          dz = Z[GW_END2[jw] - 1] - Z[GW_END1[jw] - 1];
          ltotal += LENGTH (dx, dy, dz);
          jw++;
        }
      }
      if (ih < number12[i]) {
        GW_END1[jw] = k + 1;
        GW_END2[jw] = k + number23[i] + 2;
        dx = X[GW_END2[jw] - 1] - X[GW_END1[jw] - 1];
        dy = Y[GW_END2[jw] - 1] - Y[GW_END1[jw] - 1];
        dz = Z[GW_END2[jw] - 1] - Z[GW_END1[jw] - 1];
        ltotal += LENGTH (dx, dy, dz);
        jw++;
      }
      k++;
    }

    /* Set radii for wires */
    radius = factor[i] * areaTotal / (2 * M_PI * ltotal);
    if (radius > maxradius) maxradius = radius;
    for (j = 0; j < jw; j++) GW_RAD[j] = radius;
    NodeCount = jn;
    SWireCount = jw;
  }

  /* Add unique nodes & wires to the global arrays */
  removeRedundantItems (maxradius);

} /* createWiresFromInputs */


static void removeRedundantItems (maxRadius)
    float maxRadius;
{
  int i, j, k;
  float dx, dy, dz, dd;
  extern int NodeCount, SWireCount;

  /* Remove new nodes which already exist */
  for (i = 0; i < NodeCount; i++) {
    for (j = i + 1; j < NodeCount; j++) {
      dx = X[i] - X[j];
      dy = Y[i] - Y[j];
      dz = Z[i] - Z[j];
      dd = LENGTH (dx, dy, dz);

      /* Eliminate redundant nodes */
      if (dd < maxRadius) {
        for (k = j; k < NodeCount - 1; k++) {
          X[k] = X[k + 1];
          Y[k] = Y[k + 1];
          Z[k] = Z[k + 1];
        }
        NodeCount--;

        /* Correct nodes for wires */
```

```
      for (k = 0; k < SWireCount; k++) {
       if (GW_END1[k] == j + 1) GW_END1[k] = i + 1;
       if (GW_END2[k] == j + 1) GW_END2[k] = i + 1;
       if (GW_END1[k] > j + 1) GW_END1[k] = GW_END1[k] - 1;
       if (GW_END2[k] > j + 1) GW_END2[k] = GW_END2[k] - 1;
      }
      j--;
     }
    }
   }

   /* Reallocate arrays */
   X = (float *) XtRealloc ((char *) X, sizeof (float) * NodeCount);
   Y = (float *) XtRealloc ((char *) Y, sizeof (float) * NodeCount);
   Z = (float *) XtRealloc ((char *) Z, sizeof (float) * NodeCount);

   /* Eliminate redundant & zero length wires & wires in ground plane */
   for (i = 0; i < SWireCount; i++) {
    for (j = i + 1; j < SWireCount; j++) {
     if ((GW_END1[i] == GW_END2[i]) ||
        ((GN_IPERF[0] >= 0) &&
         ((Z[GW_END1[i]] == 0) && (Z[GW_END2[i]] == 0))) ||
        (GW_END1[i] == GW_END1[j] && GW_END2[i] == GW_END2[j]) ||
        (GW_END1[i] == GW_END2[j] && GW_END2[i] == GW_END1[j])) {
      for (k = j; k < SWireCount - 1; k++) {
       GW_END1[k] = GW_END1[k + 1];
       GW_END2[k] = GW_END2[k + 1];
      }
      SWireCount--;
      j--;
     }
    }
   }

   /* Reallocate arrays */
   GW_END1 = (int *) XtRealloc ((char *) GW_END1, sizeof (int) * SWireCount);
   GW_END2 = (int *) XtRealloc ((char *) GW_END2, sizeof (int) * SWireCount);
   GW_ITG = (int *) XtRealloc ((char *) GW_ITG, sizeof (int) * SWireCount);
   GW_NS = (int *) XtRealloc ((char *) GW_NS, sizeof (int) * SWireCount);
   GW_RAD = (float *) XtRealloc ((char *) GW_RAD, sizeof (float) * SWireCount);

   /* Set tag & segment number defaults */
   for (i = oldSWireCount; i < SWireCount; i++) {
    GW_ITG[i] = 0;
    GW_NS[i] = 1;
   }
  }
```

fMeshes.c:

```
/* Filename: fMeshes.c
 * Procedures for creating the Meshes Window
 */

#include "control.h"
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/SelectioB.h>
#include <Xm/Text.h>
#include <Xm/RowColumn.h>

Widget meshesShell = NULL, meshesRowColumn, meshesIndexLabel;

extern Widget topLevel;
extern XmString *createMeshesStringTable ();
extern int MeshesCount;

/* Forward declarations for callbacks */

extern void meshesListCB ();
extern void nodeTextCB ();
extern void meshesAddButtonCB ();
extern void meshesModifyButtonCB ();
extern void meshesDeleteButtonCB ();
extern void meshesTextCB ();
extern void meshesOkButtonCB ();
extern void meshesApplyButtonCB ();
extern void meshesResetButtonCB ();
extern void cancelButtonCB ();
extern void HighlightText ();

static Widget meshesCreateSelectionBox ();
static Widget meshesCreateWorkArea ();

/*****************************************************************************/
void createMeshesWindow ()

{
```

```
Widget form, selectionBox, workArea, list;
Arg args [10];
int n = 0;
XmString string;
Position x, y;
extern void newEscapeAction();

XtTranslateCoords (topLevel, (Position) 0, (Position) 0, &x, &y);
XtSetArg (args [n], XmNx, x); n++;
XtSetArg (args [n], XmNy, y + 100); n++;
meshesShell = XtCreatePopupShell
  ("Wire Mesh Surface", topLevelShellWidgetClass, topLevel, args, n);

newEscapeAction(meshesShell);

form = XmCreateForm (meshesShell, "meshesForm", args, n);
XtManageChild (form);

/* Create index label */
n = 0;
string = XmStringCreateSimple ("Index 0 of 0");
XtSetArg (args [n], XmNlabelString, string); n++;
XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNleftOffset, 5); n++;
XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNrightOffset, 5); n++;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNtopOffset, 15); n++;
meshesIndexLabel = XmCreateLabel (form, "indexLabel", args, n);
XtManageChild (meshesIndexLabel);
XmStringFree (string);

/* Create selection box */
selectionBox =
  meshesCreateSelectionBox (form, meshesIndexLabel);
XtManageChild (selectionBox);

/* Create work area */
workArea = meshesCreateWorkArea (selectionBox);
XtManageChild (workArea);

/* Select current item in selection box */
if (MeshesCount) {
  list = XmSelectionBoxGetChild (selectionBox, XmDIALOG_LIST);
  XmListSelectPos (list, 1, TRUE);
}

} /* end createMeshesWindow */

/*****************************************************************************/
static Widget meshesCreateSelectionBox (parent, widget)

  Widget parent, widget;
{
  Widget box, child [2], list;
  Arg args [15];
  int n;
  XmString string1, string2, string3;
  char str [160];
  extern void newSelectActionTable ();

  /* Create toplevel selection box */
  n = 0;
  sprintf (str, "%10s%10s%10s%10s%12s%12s%12s\n%10s%10s%10s%10s%12s%12s%12s",
        "Node of", "Node of", "Node of","Node of", "Number  ",
        "Number  ", "Area ", "Corner1", "Corner2", "Corner3",
        "Corner4", "Corner 1-2", "Corner 2-3", "Factor");
  string1 = XmStringCreateLtoR (str, XmSTRING_DEFAULT_CHARSET);
  string2 = XmStringCreateLtoR ("Cancel", XmSTRING_DEFAULT_CHARSET);
  string3 = XmStringCreateLtoR ("Reset", XmSTRING_DEFAULT_CHARSET);
  XtSetArg (args [n], XmNshadowThickness, 1); n++;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNleftOffset, 15); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNrightOffset, 15); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNtopWidget, widget); n++;
  XtSetArg (args [n], XmNtopOffset, 10); n++;
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNbottomOffset, 15); n++;
  XtSetArg (args [n], XmNlistVisibleItemCount, 5); n++;
  XtSetArg (args [n], XmNlistLabelString, string1); n++;
  XtSetArg (args [n], XmNhelpLabelString, string2); n++;
  XtSetArg (args [n], XmNcancelLabelString, string3); n++;
  XtSetArg (args [n], XmNlistVisibleItemCount, 5); n++;
  box = XmCreateSelectionBox (parent, "meshesBox", args, n);
  XmStringFree (string1);
  XmStringFree (string2);
  XmStringFree (string3);

  /* Register callbacks for SelectionBox list. */
  list = XmSelectionBoxGetChild (box, XmDIALOG_LIST);
```

```
    n=0;
    XtSetArg(args[n], XmNselectionPolicy, XmEXTENDED_SELECT); n++;
    XtSetValues(list, args, n);
    XtAddCallback (list, XmNextendedSelectionCallback,
       meshesListCB, NULL);
    newSelectActionTable (list);

    /* Unmanage unneeded children */
    n = 0;
    child [n++] = XmSelectionBoxGetChild (box, XmDIALOG_SELECTION_LABEL);
    child [n++] = XmSelectionBoxGetChild (box, XmDIALOG_TEXT);
    XtUnmanageChildren (child, n);

    child [0] = XmSelectionBoxGetChild (box, XmDIALOG_APPLY_BUTTON);
    XtAddCallback (child [0], XmNactivateCallback,
            meshesApplyButtonCB, list);
    XtManageChild (child [0]);

    /* Add callbacks for ok, apply, reset, & cancel buttons */
    child [0] = XmSelectionBoxGetChild (box, XmDIALOG_OK_BUTTON);
    XtAddCallback (child [0], XmNactivateCallback,
            meshesOkButtonCB, list);
    child [0] = XmSelectionBoxGetChild (box, XmDIALOG_CANCEL_BUTTON);
    XtAddCallback (child [0], XmNactivateCallback,
            meshesResetButtonCB, list);

    child [0] = XmSelectionBoxGetChild (box, XmDIALOG_HELP_BUTTON);
    XtAddCallback (child [0], XmNactivateCallback, cancelButtonCB, NULL);

    /* Remove default button */
    n = 0;
    XtSetArg (args [n], XmNdefaultButton, NULL); n++;
    XtSetValues (box, args, n);

    return (box);

} /* end meshesCreateSelectionBox */

/*****************************************************************************/
static Widget meshesCreateWorkArea (parent)

    Widget parent;
{
    Widget box, rowColumn, label, corner1Text, corner2Text,
        corner3Text, corner4Text, number12Text, number23Text,
        factorText, button, mButton, list;
    Arg args [10];
    int n;
    XmString string;

    /* Create outer form box */
    n = 0;
    box = XmCreateForm (parent, "workArea", args, n);
    XtManageChild (box);

    /* Create RowColumn box */
    n = 0;
    XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNleftOffset, 10); n++;
    XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNpacking, XmPACK_COLUMN); n++;
    XtSetArg (args [n], XmNnumColumns, 4); n++;
    XtSetArg (args [n], XmNorientation, XmVERTICAL); n++;
    XtSetArg (args [n], XmNisAligned, True); n++;
    XtSetArg (args [n], XmNentryAlignment, XmALIGNMENT_END); n++;
    rowColumn = XmCreateRowColumn (box, "rowColumn", args, n);
    meshesRowColumn = rowColumn;
    XtManageChild (rowColumn);

    /* Create Node of Corner 1 label */
    n = 0;
    string = XmStringCreateLtoR ("Node of Corner 1:", XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args [n], XmNlabelString, string); n++;
    label = XmCreateLabel (rowColumn, "corner1Label", args, n);
    XtManageChild (label);
    XmStringFree (string);

    /* Create Node of Corner 2 label */
    n = 0;
    string = XmStringCreateLtoR ("Node of Corner 2:", XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args [n], XmNlabelString, string); n++;
    label = XmCreateLabel (rowColumn, "corner2Label", args, n);
    XtManageChild (label);
    XmStringFree (string);

    /* Create Node of Corner 3 label */
    n = 0;
    string = XmStringCreateLtoR ("Node of Corner 3:", XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args [n], XmNlabelString, string); n++;
    label = XmCreateLabel (rowColumn, "corner3Label", args, n);
    XtManageChild (label);
    XmStringFree (string);
```

```
/* Create Node of Corner 4 label */
n = 0;
string = XmStringCreateLtoR ("Node of Corner 4:", XmSTRING_DEFAULT_CHARSET);
XtSetArg (args [n], XmNlabelString, string); n++;
label = XmCreateLabel (rowColumn, "corner4Label", args, n);
XtManageChild (label);
XmStringFree (string);

/* Create Node of Corner 1 text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNmaxLength, 1); n++;
corner1Text = XmCreateText (rowColumn, "corner1Text", args, n);
XtManageChild (corner1Text);

/* Create Node of Corner 2 text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNmaxLength, 6); n++;
corner2Text = XmCreateText (rowColumn, "corner2Text", args, n);
XtManageChild (corner2Text);

/* Create Node of Corner 3 text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNmaxLength, 11); n++;
corner3Text = XmCreateText (rowColumn, "corner3Text", args, n);
XtManageChild (corner3Text);

/* Create Node of Corner 4 text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNmaxLength, 11); n++;
corner4Text = XmCreateText (rowColumn, "corner4Text", args, n);
XtManageChild (corner4Text);

/* Create Number of Patches from Corner 1 to 2 label */
n = 0;
string = XmStringCreateLtoR ("Number of Wires\nCorner 1 to 2:",
                    XmSTRING_DEFAULT_CHARSET);
XtSetArg (args [n], XmNlabelString, string); n++;
label = XmCreateLabel (rowColumn, "number12Label", args, n);
XtManageChild (label);
XmStringFree (string);

/* Create Number of Patches from Corner 2 to 3 label */
n = 0;
string = XmStringCreateLtoR ("Number of Wires\nCorner 2 to 3:",
                    XmSTRING_DEFAULT_CHARSET);
XtSetArg (args [n], XmNlabelString, string); n++;
label = XmCreateLabel (rowColumn, "number23Label", args, n);
XtManageChild (label);
XmStringFree (string);

/* Create Area Factor label */
n = 0;
string = XmStringCreateLtoR ("Area Factor:", XmSTRING_DEFAULT_CHARSET);
XtSetArg (args [n], XmNlabelString, string); n++;
label = XmCreateLabel (rowColumn, "factorLabel", args, n);
XtManageChild (label);
XmStringFree (string);

/* Create dummy label */
n = 0;
label = XmCreateLabel (rowColumn, " ", args, n);
XtManageChild (label);

/* Create Number of Patches from Corner 1 to 2 text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNmaxLength, 11); n++;
number12Text = XmCreateText (rowColumn, "number12Text", args, n);
XtManageChild (number12Text);

/* Create Number of Patches from Corner 2 to 3 text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNmaxLength, 11); n++;
number23Text = XmCreateText (rowColumn, "number23Text", args, n);
XtManageChild (number23Text);

/* Create Area Factor text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNmaxLength, 11); n++;
factorText = XmCreateText (rowColumn, "factorText", args, n);
```

```
    XtManageChild (factorText);

    /* Add callbacks for all text fields */
    XtAddCallback (corner1Text, XmNactivateCallback, meshesTextCB,
            corner2Text);
    XtAddCallback (corner1Text, XmNfocusCallback, HighlightText, NULL);
    XtAddCallback (corner2Text, XmNactivateCallback, meshesTextCB,
            corner3Text);
    XtAddCallback (corner2Text, XmNfocusCallback, HighlightText, NULL);
    XtAddCallback (corner3Text, XmNactivateCallback, meshesTextCB,
            corner4Text);
    XtAddCallback (corner3Text, XmNfocusCallback, HighlightText, NULL);
    XtAddCallback (corner4Text, XmNactivateCallback, meshesTextCB,
            number12Text);
    XtAddCallback (corner4Text, XmNfocusCallback, HighlightText, NULL);
    XtAddCallback (number12Text, XmNactivateCallback, meshesTextCB,
            number23Text);
    XtAddCallback (number12Text, XmNfocusCallback, HighlightText, NULL);
    XtAddCallback (number23Text, XmNactivateCallback, meshesTextCB,
            factorText);
    XtAddCallback (number23Text, XmNfocusCallback, HighlightText, NULL);

    /* Create Add button.  Put in dummy string of 6 characters so that
     * this button will be the same size as the others.  Then reset
     * label string.
     */
    n = 0;
    string = XmStringCreateSimple ("xxxxxx");
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetArg (args [n], XmNleftAttachment, XmATTACH_WIDGET); n++;
    XtSetArg (args [n], XmNleftWidget, rowColumn); n++;
    XtSetArg (args [n], XmNleftOffset, 10); n++;
    XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNrightOffset, 10); n++;
    button = XmCreatePushButton (box, "addButton", args ,n);
    XtManageChild (button);
    XmStringFree (string);

    XtAddCallback (factorText, XmNactivateCallback, meshesTextCB, button);
    XtAddCallback (factorText, XmNfocusCallback, HighlightText, NULL);

    string = XmStringCreateSimple ("Add");
    XtSetArg (args [n], XmNrecomputeSize, FALSE); n++;
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetValues (button, args, n);
    XmStringFree (string);

    /* Register callback for add button.  */
    list = XmSelectionBoxGetChild (parent, XmDIALOG_LIST);
    XtAddCallback (button, XmNactivateCallback, meshesAddButtonCB, list);

    /* Create Modify button */
    n = 0;
    string = XmStringCreateSimple ("Modify");
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNrightOffset, 10); n++;
    XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
    XtSetArg (args [n], XmNtopWidget, button); n++;
    XtSetArg (args [n], XmNtopOffset, 10); n++;
    mButton = XmCreatePushButton (box, "modifyButton", args ,n);
    XtManageChild (mButton);
    XmStringFree (string);

    XtAddCallback (mButton, XmNactivateCallback, meshesModifyButtonCB, list);

    /* Create Delete button */
    n = 0;
    string = XmStringCreateSimple ("Delete");
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNrightOffset, 10); n++;
    XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
    XtSetArg (args [n], XmNtopWidget, mButton); n++;
    XtSetArg (args [n], XmNtopOffset, 10); n++;
    button = XmCreatePushButton (box, "deleteButton", args ,n);
    XtManageChild (button);
    XmStringFree (string);

    XtAddCallback (button, XmNactivateCallback, meshesDeleteButtonCB, list);
    return (box);

} /* end meshesCreateWorkArea */
```

A.9 fDescrip.c

fDescrip.c:

```
/*
 * Filename : fDescrip.c
 *
 * Procedures for creating the Description Summary window.
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/types.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/PanedW.h>
#include <Xm/Text.h>
#include "actionArea.h"
#include "cFileMenu.h"

#define min(x,y) ((x < y) ? (x) : (y))
#define max(x,y) ((x > y) ? (x) : (y))

extern Widget topLevel;

Widget descripShell = NULL;
Widget descripText;
char *freqUnits [] = {"KHz", "MHz", "GHz"};

static char lineFeed [] = "\n\n";

static void closeButtonCB ();
static void describeElectrical ();
static void describeGeometry ();
static void describeSolution ();
static void destroyCB ();
static void insertDescripText ();
static void printButtonCB ();
int loHiFreq ();

extern Boolean isPoppedUp ();

/*****************************************************************************/
void openDescripWindow ()
{
 Widget pane, actionA;
 Arg args [10];
 static ActionAreaItem actionItems[] = {
    {"Print",  printButtonCB,   NULL},
    {"Close",  closeButtonCB,   NULL},
 };
 extern void newEscapeAction();

 if (descripShell != NULL && !isPoppedUp(descripShell)) {
  XtDestroyWidget (descripShell);
  descripShell = NULL;
 }

 if (descripShell != NULL) return;

 /* Build the text window */
 descripShell = XtVaCreatePopupShell (NULL,
    topLevelShellWidgetClass, topLevel,
    XmNtitle,          "Description Summary",
    XmNallowShellResize,  True,
    XmNdeleteResponse,   XmDESTROY,
    NULL);

 newEscapeAction(descripShell);

 XtSetArg (args[0], XmNsashWidth, 1);
 XtSetArg (args[1], XmNsashHeight, 1);
 pane = XmCreatePanedWindow (descripShell, "pane", args, 2);
 XtAddCallback (pane, XmNdestroyCallback, destroyCB, NULL);
 XtManageChild (pane);

 XtSetArg (args[0], XmNrows, 20);
 XtSetArg (args[1], XmNcolumns, 80);
 XtSetArg (args[2], XmNeditable, False);
 XtSetArg (args[3], XmNblinkRate, 0);
 XtSetArg (args[4], XmNcursorPositionVisible, False);
 XtSetArg (args[5], XmNeditMode, XmMULTI_LINE_EDIT);
 XtSetArg (args[6], XmNwordWrap, True);
 XtSetArg (args[7], XmNscrollHorizontal, False);
 descripText = XmCreateScrolledText (pane, "text", args, 8);
 XtManageChild (descripText);

 /* Create the action area */
 actionA = createActionArea (pane, actionItems, XtNumber (actionItems));
```

```
  XtPopup (descripShell, XtGrabNone);

  insertDescripText (descripText);

 } /* end createDescripWindow */

 /*****************************************************************
  * Prints output to default printer.
  *****************************************************************/

 static void printButtonCB (void)
 {
  FILE *fp;
  char *string, filename [20], command [132];
  extern FILE *efopen ();

  /* Create unique filename then open the file */
  tmpnam (filename);
  if ((fp = efopen (filename, "w")) == NULL)
   return;

  /* Write the text string to the file and print it */
  string = XmTextGetString (descripText);
  fprintf (fp, "%s", string);
  fclose (fp);
  sprintf (command, "%s %s", getenv ("MOM_PRINT_TEXT"), filename);
  system (command);

  remove (filename);
  XtFree (string);

 } /* end printButtonCB */

 /*****************************************************************/
 static void closeButtonCB (void)
 {
  XtDestroyWidget (descripShell);

 } /* end closeButtonCB */

 /*****************************************************************/
 static void destroyCB (void)
 {
  descripShell = NULL;

 } /* end destroyCB */

 /*****************************************************************/

 static void insertDescripText (text_w)
  Widget text_w;
 {
  Boolean freeMem = True;
  char *string, text [100];
  static char noFilename [] = "untitled";
  XmTextPosition position = 0;
  time_t currentTime;
  extern Widget inputFilenameText;
  extern Widget necInputFileText;
  extern char *inputFilename;
  extern char *necInputFilename;
  int i, nCM = 1, LEN = 160;
  char *str = NULL;

  /* Output filename */
  string = inputFilename;
  if (string == NULL || strlen (string) == 0) {
   string = necInputFilename;
   if (string == NULL || strlen (string) == 0) {
    string = noFilename;
    freeMem = False;
   }
  }

  sprintf (text, "FILENAME: %s\n", string);
  XmTextInsert (text_w, position, text);
  position += strlen (text);

  /* Output current date and time */
  time (&currentTime);
  ascftime (text, "DATE: %d %b %Y\nTIME: %T\n",
                  localtime (&currentTime));
  XmTextInsert (text_w, position, text);
  position += strlen (text);

  /* Comments */
  if (CommentCount) {
   str = XtMalloc(LEN);
   bzero(str, LEN);
   for (i = 0; i < CommentCount; i++) {
    str = XtRealloc(str, nCM * LEN);
    strcat(str, CM[i].line);
```

```
      str[strlen(str)] = '\n';
      str[strlen(str) + 1] = '\0';
      nCM++;
    }
    str[strlen(str)] = '\0';
    XmTextInsert (text_w, position, lineFeed);
    position += 2;
    sprintf (text, "%s\n", "COMMENTS:");
    XmTextInsert (text_w, position, text);
    position += strlen (text);
    XmTextInsert (text_w, position, str);
    position += strlen (str);
    XtFree(str);
  }

  describeGeometry (text_w, &position);
  describeElectrical (text_w, &position);
  describeSolution (text_w, &position);
}

/*****************************************************************************/

static void describeGeometry (text_w, position)
  Widget text_w;
  XmTextPosition *position;
{
  char text [81];
  int i, count, patches, wireUnknowns = 0;
  static char *dimensions [] = {"Meters", "Centimeters", "Feet", "Inches"};
  static char *environments [] = {"Free Space", "Ground Plane"};
  extern int DimIndex, EnvIndex;

  /* Geometry Description */
  sprintf (text, "\n\n%s\n\n", "GEOMETRY DESCRIPTION:");
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
  sprintf (text, "Dimension: %s\n", dimensions[DimIndex]);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
  sprintf (text, "Environment: %s\n", environments[EnvIndex]);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);

  if  (SWireCount || TaperWireCount || CWireCount || WireArcCount ||
       HelixOrSpiralCount) {
    sprintf (text, "\n%-20s%-20s%s\n", "Wire Types", "Number of Wires",
          "Number of Segments");
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
  }

  /* Straight Wires & Segments */
  if (SWireCount > 0) {
    count = 0;
    for (i=0; i<SWireCount; i++)
      count += GW_NS[i];
    sprintf (text, "%-20s%8d%22d\n", "Straight Wires", SWireCount, count);
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
    wireUnknowns += count;
  }

  /* Tapered Wires & Segments */
  if (TaperWireCount > 0) {
    count = 0;
    for (i=0; i<TaperWireCount; i++) {
      if (GC_IX[i] == 2) {
                float wireLength, RD, x, y, z;
                /* Compute wire length */
                x = X [GC_END1 [i]] - X [GC_END2 [i]];
                x = x * x;
                y = Y [GC_END1 [i]] - Y [GC_END2 [i]];
                y = y * y;
                z = Z [GC_END1 [i]] - Z [GC_END2 [i]];
                z = z * z;
                wireLength = (float) sqrt (x + y + z);
                RD = (wireLength - GC_DEL1[i]) / (wireLength - GC_DEL2[i]);
                count += 1 + (log (GC_DEL1[i] / GC_DEL2[i]) / log (RD));
      } else
        count += GC_NS[i];
    }
    sprintf (text, "%-20s%8d%22d\n", "Tapered Wires", TaperWireCount, count);
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
    wireUnknowns += count;
  }

  if (CWireCount > 0) {
    count = 0;
    for (i=0; i<CWireCount; i++)
      count += CW_NS[i];
    sprintf (text, "%-20s%8d%22d\n", "Cantenary Wires", CWireCount, count);
    XmTextInsert (text_w, *position, text);
```

```
    *position += strlen (text);
    wireUnknowns += count;
  }

  /* Wire Arcs & Segments */
  if (WireArcCount > 0) {
    count = 0;
    for (i=0; i<WireArcCount; i++)
      count += GA_NS[i];
    sprintf (text, "%-20s%8d%22d\n", "Wire Arc", WireArcCount, count);
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
    wireUnknowns += count;
  }

  /* Helix or Spiral Wires & Segments */
  if (HelixOrSpiralCount > 0) {
    count = 0;
    for (i=0; i<HelixOrSpiralCount; i++)
      count += GH_NS[i];
    sprintf (text, "%-20s%8d%22d\n", "Helix or Spiral",
                HelixOrSpiralCount, count);
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
    wireUnknowns += count;
  }

  /* Meshes - not implemented in this phase */

  /* Number of Surface Patches */
  patches = 0;
  if (MultiplePatchCount > 0) {
    for (i=0; i<MultiplePatchCount; i++)
      patches += SM_Number12[i] * SM_Number23[i];
  }
  patches += SurfacePatchCount;
  sprintf (text, "\nNumber of Surface Patches = %d\n\n", patches);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);

  /* Number of Unknowns */
  sprintf (text, "Number of Wire Unknowns = %d\n", wireUnknowns);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
  sprintf (text, "Number of Patch Unknowns = %d\n", patches * 2);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
  sprintf (text, "Total Number of Unknowns = %d\n\n",
              wireUnknowns + (patches * 2));
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);

  /* Number of Rotations, Reflections & Transformations */
  if (RotationCount > 0) {
    count = 0;
    for (i=0; i<RotationCount; i++)
      count += GR_NR[i];
    sprintf (text, "Number of Rotations = %d\n", count);
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
  }
  if (TransformCount > 0) {
    count = 0;
    for (i=0; i<TransformCount; i++)
      count += GM_NRPT[i];
    sprintf (text, "Number of Transformations = %d\n", count);
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
  }
  if (ReflectionCount > 0) {
    count = 0;
    for (i=0; i<ReflectionCount; i++)
      count |= GX_IXYZ[i];
    sprintf (text, "Reflections:\n");
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
    sprintf (text, "   x-axis = %s\n", count & 0x1 ? "YES" : "NO");
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
    sprintf (text, "   y-axis = %s\n", count & 0x2 ? "YES" : "NO");
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
    sprintf (text, "   z-axis = %s\n", count & 0x4 ? "YES" : "NO");
    XmTextInsert (text_w, *position, text);
    *position += strlen (text);
  }
} /* end describeGeometry */

/*****************************************************************************/

static void describeElectrical (text_w, position)
  Widget text_w;
  XmTextPosition *position;
```

```
{
 char text [81];
 int i, count;
 extern int FrequencyIndex;

 sprintf (text, "\n\nELECTRICAL DESCRIPTION:\n\n");
 XmTextInsert (text_w, *position, text);
 *position += strlen (text);

 /* Number of Frequencies */
 count = 0;
 for (i=0; i<FrequencyCount; i++)
  count += FR_NFRQ[i];
 sprintf (text, "Number of Frequencies = %d\n", count);
 XmTextInsert (text_w, *position, text);
 *position += strlen (text);

 /* Lowest & Highest Frequency */
 if (FrequencyCount > 0) {
  float lo, hi;

  loHiFreq (&lo, &hi);
  sprintf (text, "Lowest Frequency = %g %s\n", lo,
                 freqUnits[FrequencyIndex]);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
  sprintf (text, "Highest Frequency = %g %s\n", hi,
                 freqUnits[FrequencyIndex]);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 if (LoadsCount || VoltageSourcesCount || IncidentPlaneWaveCount ||
    TransmissionLinesCount || TwoPortNetsCount || InsulatedWiresCount) {
  sprintf (text, "\n%31s\n", "Number");
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 /* Number of Loads */
 if (LoadsCount > 0) {
  sprintf (text, "%-25s%4d\n", "Loads", LoadsCount);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 /* Number of Voltage Sources */
 if (VoltageSourcesCount > 0) {
  sprintf (text, "%-25s%4d\n", "Voltage Sources", VoltageSourcesCount);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 /* Number of Incident Plane Waves */
 if (IncidentPlaneWaveCount > 0) {
  sprintf (text, "%-25s%4d\n", "Incident Plane Waves",
                 IncidentPlaneWaveCount);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 /* Number of Transmission Lines */
 if (TransmissionLinesCount > 0) {
  sprintf (text, "%-25s%4d\n", "Transmission Lines",
                 TransmissionLinesCount);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 /* Number of Two Port Networks */
 if (TwoPortNetsCount > 0) {
  sprintf (text, "%-25s%4d\n", "Two Port Networks",
                 TwoPortNetsCount);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 /* Number of Insulated Wires */
 if (InsulatedWiresCount > 0) {
  sprintf (text, "%-25s%4d\n", "Insulated Wires",
                 InsulatedWiresCount);
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
 }

 /* Ground type */
 if (GroundParamCount > 0)
 if ((GN_IPERF[0] == 0) || (GN_IPERF[0] == 2)) {
  sprintf (text, "\nGround Type: %s",
                 GN_IPERF[0] ? "Sommerfeld/Asymptotic method" :
                 "reflection-coefficient approximation");
  XmTextInsert (text_w, *position, text);
  *position += strlen (text);
```

```
    }
  } /* end describeElectrical */

  /*****************************************************************************/

  static void describeSolution (text_w, position)
    Widget text_w;
    XmTextPosition *position;
  {
    int i, nearElectric, nearMagnetic, radPattern;
    char text [81];

    nearElectric = 0;
    for (i = 0; i < NearElectricCount; i++)
      nearElectric += NE_NRX[i] * NE_NRY[i] * NE_NRZ[i];
    nearMagnetic = 0;
    for (i = 0; i < NearMagneticCount; i++)
      nearMagnetic += NH_NRX[i] * NH_NRY[i] * NH_NRZ[i];
    radPattern = 0;
    for (i = 0; i < RadiationPatternCount; i++)
      radPattern += RP_NPH[i] * RP_NTH[i];

    if (MaxCouplingCount || nearElectric || nearMagnetic || radPattern) {
      sprintf (text, "\n\nSOLUTION DESCRIPTION:\n\n");
      XmTextInsert (text_w, *position, text);
      *position += strlen (text);
      sprintf (text, "%41s\n", "Number");
      XmTextInsert (text_w, *position, text);
      *position += strlen (text);
      if (MaxCouplingCount) {
        sprintf (text, "%-30s%9d\n", "Maximum Coupling Calculation",
                 MaxCouplingCount);
        XmTextInsert (text_w, *position, text);
        *position += strlen (text);
      }
      if (nearElectric) {
        sprintf (text, "%-30s%9d\n", "Near Electric Field", nearElectric);
        XmTextInsert (text_w, *position, text);
        *position += strlen (text);
      }
      if (nearMagnetic) {
        sprintf (text, "%-30s%9d\n", "Near Magnetic Field", nearMagnetic);
        XmTextInsert (text_w, *position, text);
        *position += strlen (text);
      }
      if (radPattern) {
        sprintf (text, "%-30s%9d\n", "Radiation Pattern", radPattern);
        XmTextInsert (text_w, *position, text);
        *position += strlen (text);
      }
    }
  } /* end describeSolution */

  /*****************************************************************************
   * Returns the lowest and highest frequencies.  Function returns 0 if
   * FrequencyCount is not greater than zero.  Returns 1, otherwise.
   *****************************************************************************/

  int loHiFreq (lo, hi)
    float *lo, *hi;
  {
    int returnValue = 0, i;
    float freq;

    if (FrequencyCount > 0) {

      *lo = *hi = FR_FMHZ[0];
      for (i=0; i<FrequencyCount; i++) {

        /* linear stepping */
        if (FR_IFRQ[i] == 0)
          freq = FR_FMHZ[i] + FR_DELFRQ[i] * (FR_NFRQ[i] - 1);

        /* Multiplicative stepping */
        else
          freq = FR_FMHZ[i] * pow ((double) FR_DELFRQ[i],
                                   (double) FR_NFRQ[i] - 1);
        *lo = min (*lo, freq);
        *hi = max (*hi, freq);

        returnValue = 1;
      }
    }
    return (returnValue);
  } /* end loHiFreq */
```

A.10 cDiagnostics.c, fDiagnostics.c:

cDiagnostics.c:

```
/* cDiagnostics.c
 *
 * Callbacks for the Diagnostics window
 */

#include "control.h"
#include "cFileMenu.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <X11/IntrinsicP.h>
#include <X11/ShellP.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/PanedW.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>

extern Widget topLevel;

/* Diagnostics Window Widgets */

extern Widget diagnosticsShell,
        diagIndividualWires,    /* Check boxes for Geometry */
        diagWireJunctions,      /*  Diagnostic options */
        diagCrossedWires,
        diagIndividualPatches,
        diagPatchWireJunctions,
        diagWsegmentLength,     /* Text boxes for geometry */
        diagWsegRadiusRatio1,   /*  guidelines */
        diagWradius,
        diagWsegLengthRatio,
        diagWradiusRatio,
        diagWedgeLength,
        diagWedgeSegRatio,
        diagWedgeRadiusRatio,
        diagEsegmentLength,
        diagEsegRadiusRatio1,
        diagEradius,
        diagEsegLengthRatio,
        diagEradiusRatio,
        diagEwireRadii,
        diagEedgeLength,
        diagEedgeSegRatio,
        diagEedgeRadiusRatio,
        diagGeometry,           /* Radio buttons for Diagnostic */
        diagElectrical,         /*  type */
        diagText;               /* Text widget for diagnostics */

extern int NumSegs;
extern int *Jwire;
int *Idiag = NULL;              /* Each element tells necDisplay what
                                 * color to use for each segment */

float guidelineWarnings [9], guidelineErrors [9];

static float *sSegLength = NULL,    /* Straight wire segment lengths */
        *tSegLength = NULL,     /* Tapered wire segment lengths */
        *cSegLength = NULL,     /* Catenary wire segment lengths */
        *wSegLength = NULL,     /* Wire Arc segment lengths */
        *hSegLength = NULL,     /* Helix or Spiral segment lengths */
        *taperSeg1 = NULL,      /* 1st and last tapered segment */
        *taperSeg2 = NULL,
        *tLengthRad = NULL,     /* Minimum length/radius for tapers */
        *hLengthRad = NULL;     /* Minimum length/radius for helix */
static float lowWavelength;     /* Shortest wavelength */
static XmTextPosition lastPosition;  /* Current position in diagText */

static enum WireTypes {Straight, Tapered, Catenary, WireArc, HelixSpiral};
static char *wireNames [] = {"Straight", "Tapered", "Catenary", "Wire Arc",
                             "Helix/Spiral"};

static void catenaryWireSegmentLengths ();
static void catsol ();
static void catexp ();
static void checkRadiusRatios ();
static void checkRadiusToWavelength ();
static void checkSegLengthRatios ();
static void checkSegLengthToWavelength ();
static void checkSegLengthToRadius ();
static void findCoincidentWires ();
static void findCrossedWires ();
static void findMatchPointErrors ();
static void runElectricalDiagnostics ();
static void runGeometryDiagnostics ();
```

```
static void runIndividualWiresDiagnostics ();
static void runWireJunctionsDiagnostics ();
static void calculateSegmentLengths ();
static void taperWireSegmentLengths ();
static void wireArcSegmentLengths ();
static void helixSpiralSegmentLengths ();

/* Pointer into wireErrors for indicting the errors of wires */
unsigned long *errors;
unsigned long *wireErrors = NULL;

int loHiFreq ();

/*****************************************************************
 * Frees memory associated with diagnostics
 *****************************************************************/

void diagDestroyCB (void)
{
  XtFree ((char *) sSegLength);
  XtFree ((char *) tSegLength);
  XtFree ((char *) cSegLength);
  XtFree ((char *) taperSeg1);
  XtFree ((char *) taperSeg2);
  XtFree ((char *) tLengthRad);
  XtFree ((char *) wSegLength);
  XtFree ((char *) hSegLength);
  XtFree ((char *) hLengthRad);
  XtFree ((char *) Idiag);
  XtFree ((char *) wireErrors);
  sSegLength = NULL;
  tSegLength = NULL;
  cSegLength = NULL;
  taperSeg1 = NULL;
  taperSeg2 = NULL;
  tLengthRad = NULL;
  wSegLength = NULL;
  hSegLength = NULL;
  hLengthRad = NULL;
  Idiag = NULL;
  wireErrors = NULL;

} /* end diagDestroyCB */

/*****************************************************************
 * Prints output to default printer.
 *****************************************************************/

void diagPrintButtonCB (void)
{
  FILE *fp;
  char *string, filename [20], command [132];
  extern FILE *efopen ();

  /* Create unique filename then open the file */
  tmpnam (filename);
  if ((fp = efopen (filename, "w")) == NULL)
    return;

  /* Write the text string to the file and print it */
  string = XmTextGetString (diagText);
  fprintf (fp, "%s", string);
  fclose (fp);
  sprintf (command, "%s %s", getenv ("MOM_PRINT_TEXT"), filename);
  system (command);

  remove (filename);
  XtFree (string);

} /* end printButtonCB */

/*****************************************************************
 * Callback for "visualize" button.
 *****************************************************************/

void diagVisualizeButtonCB (void)
{
  extern void geometryFilter ();
  extern void necDisplay ();
  extern char *necInputFilename;
  int type, source;
                int i;
                int wireNumber;
  float freq;
  type = 0;
  source = 0;
  freq = 0.0;

  /* Run geometryFilter to calculate data needed for geomtry */
  geometryFilter ();

  /* now set up Idiag array to describe wire segments color */
  Idiag = (int *) XtRealloc ((char *) Idiag, sizeof(int)*NumSegs);
```

```
  for (i = 0; i < NumSegs; i++) {
    wireNumber = Jwire[i]-1;
    /* check to see if there are any warnings and/or errors on this wire */
    if (wireErrors[wireNumber] != 0) {
      if ((wireErrors[wireNumber] >> 5) != 0)
        /* errors */
        Idiag[i] = 5;
      else
        /* warnings */
        Idiag[i] = 4;
    } else
      /* ok */
      Idiag[i] = 6;
  }
  necDisplay (necInputFilename, type, source, freq);

} /* end diagVisualizeButtonCB */

/*****************************************************************************
 * Callback for "run" button.  Determines which kind of diagnostic that
 * the user wants to run, then executes it.
 *****************************************************************************/

void diagRunButtonCB (void)
{
  XtFree ((char *) wireErrors);
  /* Allocate memory for storing errors & warnings associated with straight &
   * tapered wires */
  wireErrors = (unsigned long *) XtCalloc ((SWireCount+TaperWireCount),
                                  sizeof(unsigned long));
  if (XmToggleButtonGetState (diagGeometry))
    runGeometryDiagnostics ();
  else
    runElectricalDiagnostics ();

} /* end diagRunButtonCB */

/*****************************************************************************
 * Write diagnostics to text box
 *****************************************************************************/

static void runGeometryDiagnostics ()
{
  char string [132];
  int numWires;
  float lo, hi;
  extern char *freqUnits [];
  extern int FrequencyIndex;
  extern float FrequenciesScale [];

  XmTextSetString (diagText, "GEOMETRY DIAGNOSTICS\n");
  lastPosition = XmTextGetLastPosition (diagText);

  /* Output lowest & highest frequencies */
  if (loHiFreq (&lo, &hi)) {
    sprintf (string, "Lowest Frequency   = %g %s\n", lo,
          freqUnits [FrequencyIndex]);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);
    sprintf (string, "Highest Frequency  = %g %s\n", hi,
          freqUnits [FrequencyIndex]);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);

    /* Output shortest wavelength */
    lowWavelength = (299.8 / hi) * FrequenciesScale [FrequencyIndex];
    sprintf (string, "Shortest wavelength = %.6e meters\n", lowWavelength);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);
  }

  /* Output number of wires */
  numWires = SWireCount + TaperWireCount + CWireCount + WireArcCount +
    HelixOrSpiralCount;
  sprintf (string, "Number of wires    = %d\n\n", numWires);
  XmTextInsert (diagText, lastPosition, string);
  lastPosition += strlen (string);

  calculateSegmentLengths (&sSegLength, SWireCount, GW_NS, GW_END1, GW_END2);
  calculateSegmentLengths (&tSegLength, TaperWireCount,
                GC_NS, GC_END1, GC_END2);
  calculateSegmentLengths (&cSegLength, CWireCount, CW_NS, CW_END1, CW_END2);

  taperWireSegmentLengths ();
  catenaryWireSegmentLengths ();
  wireArcSegmentLengths ();
  helixSpiralSegmentLengths ();

  if (XmToggleButtonGetState (diagIndividualWires))
    runIndividualWiresDiagnostics ();

  if (XmToggleButtonGetState (diagWireJunctions))
    runWireJunctionsDiagnostics ();
```

```
  if (XmToggleButtonGetState (diagCrossedWires))
    findCrossedWires ();
}

/*****************************************************************
 * Diagnostics for Individual Wires
 *****************************************************************/

static void runIndividualWiresDiagnostics ()
{
float errorValue, warningValue;
char *text;

 /* Check segment length to wavelength */
 text = XmTextGetString (diagEsegmentLength);
 errorValue = atof (text);
 XtFree (text);
 text = XmTextGetString (diagWsegmentLength);
 warningValue = atof (text);
 XtFree (text);
 checkSegLengthToWavelength (errorValue, warningValue, sSegLength,
                SWireCount, "Straight");
 checkSegLengthToWavelength (errorValue, warningValue, tSegLength,
                TaperWireCount, "Tapered");
 checkSegLengthToWavelength (errorValue, warningValue, cSegLength,
                CWireCount, "Catenary");
 checkSegLengthToWavelength (errorValue, warningValue, hSegLength,
                HelixOrSpiralCount, "Helix or Spiral");
 checkSegLengthToWavelength (errorValue, warningValue, wSegLength,
                WireArcCount, "Wire Arc");

 /* Check segment length to radius */
 text = XmTextGetString (diagEsegRadiusRatio1);
 errorValue = atof (text);
 XtFree (text);
 text = XmTextGetString (diagWsegRadiusRatio1);
 warningValue = atof (text);
 XtFree (text);
 checkSegLengthToRadius (errorValue, warningValue, sSegLength,
                SWireCount, "Straight", GW_RAD);
 checkSegLengthToRadius (errorValue, warningValue, tLengthRad,
                TaperWireCount, "Tapered", NULL);
 checkSegLengthToRadius (errorValue, warningValue, cSegLength,
                CWireCount, "Catenary", CW_RAD);
 checkSegLengthToRadius (errorValue, warningValue, hLengthRad,
                HelixOrSpiralCount, "Helix or Spiral", NULL);
 checkSegLengthToRadius (errorValue, warningValue, wSegLength,
                WireArcCount, "Wire Arc", GA_RAD);

 /* Check radius to wavelength */
 text = XmTextGetString (diagEradius);
 errorValue = atof (text);
 XtFree (text);
 text = XmTextGetString (diagWradius);
 warningValue = atof (text);
 XtFree (text);
 checkRadiusToWavelength (errorValue, warningValue, GW_RAD, NULL,
                SWireCount, "Straight");
 checkRadiusToWavelength (errorValue, warningValue, GC_RAD1, GC_RAD2,
                TaperWireCount, "Tapered", NULL);
 checkRadiusToWavelength (errorValue, warningValue, CW_RAD, NULL,
                CWireCount, "Catenary");
 checkRadiusToWavelength (errorValue, warningValue, GH_WR1, GH_WR2,
                HelixOrSpiralCount, "Helix or Spiral");
 checkRadiusToWavelength (errorValue, warningValue, GA_RAD, NULL,
                WireArcCount, "Wire Arc");

} /* end runIndividualWiresDiagnostics */

/*****************************************************************
 * Diagnostics for Wire Junctions
 *****************************************************************/

static void runWireJunctionsDiagnostics ()
{
 int i;
 float **minSegs, **maxRadii;

 /* Check if any of the wires are coincident.  (Two different wires
  * are connected to the same set of nodes.
  */
 findCoincidentWires ();

 /* Allocate memory to store smallest segment & largest radius for
  * each node.  These arrays will be used to calculate Match point
  * errors.  Each array index will store three elements: value,
  * wire type & wire index.
  */
 minSegs = (float **) XtMalloc (sizeof (float *) * NodeCount);
 maxRadii = (float **) XtMalloc (sizeof (float *) * NodeCount);
 for (i = 0; i < NodeCount; i++) {
  minSegs[i] = (float *) XtMalloc (sizeof (float) * 3);
```

```
    maxRadii[i] = (float *) XtMalloc (sizeof (float) * 3);
  }
  checkSegLengthRatios (minSegs);
  checkRadiusRatios (maxRadii);
  findMatchPointErrors (maxRadii, minSegs);

  /* Free memory */
  for (i = 0; i < NodeCount; i++) {
    XtFree ((char *) minSegs[i]);
    XtFree ((char *) maxRadii[i]);
  }
  XtFree ((char *) minSegs);
  XtFree ((char *) maxRadii);

} /* end runWireJunctionsDiagnostics */

/*****************************************************************************
 * Check if any of the wires are coincident.  In other words, two
 * different wires are conected to the same set of nodes.
 *****************************************************************************/

static void findCoincidentWires ()
{
  int i, j, k, *end1, *end2, count;
  char *wireType, string [81];
  static char sType [] = "Straight";
  static char tType [] = "Tapered";
  Boolean header = True;

  /* Compare straight & tapered wires against themselves */
  for (i = 0; i < 2; i++) {
    switch (i) {
      case 0:
        end1 = GW_END1;
        end2 = GW_END2;
        count = SWireCount;
        wireType = sType;
        break;
      case 1:
        end1 = GC_END1;
        end2 = GC_END2;
        count = TaperWireCount;
        wireType = tType;
        break;
    }
    for (j = 0; j < count; j++) {
      for (k = j + 1; k < count; k++) {
        if (((end1[j] == end1[k]) && (end2[j] == end2[k])) ||
            ((end2[j] == end1[k]) && (end1[j] == end2[k]))) {
          if (header) {
            sprintf (string, "\nCoincident wires:\n");
            XmTextInsert (diagText, lastPosition, string);
            lastPosition += strlen (string);
            header = False;
          }
          if (wireType == sType) {
            wireErrors[j] |= CoincidentWireError;
            wireErrors[k] |= CoincidentWireError;
          }
          if (wireType == tType) {
            wireErrors[SWireCount + j] |= CoincidentWireError;
            wireErrors[SWireCount + k] |= CoincidentWireError;
          }
          sprintf (string, "%s %d %s %d\n", wireType, j+1, wireType, k+1);
          XmTextInsert (diagText, lastPosition, string);
          lastPosition += strlen (string);
        }
      }
    }
  }

  /* Compare straight wires against tapered wires */
  for (i = 0; i < SWireCount; i++) {
    for (j = 0; j < TaperWireCount; j++) {
      if (((GW_END1[i] == GC_END1[j]) && (GW_END2[i] == GC_END2[j])) ||
          ((GW_END1[i] == GC_END2[j]) && (GW_END2[i] == GC_END1[j]))) {
        if (header) {
          sprintf (string, "\nCoincident wires:\n");
          XmTextInsert (diagText, lastPosition, string);
          lastPosition += strlen (string);
          header = False;
        }
        wireErrors[i] |= CoincidentWireError;
        wireErrors[SWireCount + j] |= CoincidentWireError;
        sprintf (string, "straight %d tapered %d\n", i+1, j+1);
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
      }
    }
  }
} /* end findCoincidentWires */

/*****************************************************************************
```

```
 * Calculate segment lengths
 ****************************************************************************/

static void calculateSegmentLengths (length, count, numSegs, end1, end2)
    float **length;          /* Array of segment lengths */
    int count,               /* number of wires */
      *numSegs,              /* array of number of segments */
      *end1,                 /* array of start nodes */
      *end2;                 /* array of end nodes */
{
  int i, index1, index2;
  float xLength, yLength, zLength;

  /* Calculates only if count is greater than zero */
  if (count) {
    *length = (float *) XtRealloc ((char *) *length, sizeof (float) * count);
    for (i = 0; i < count; i++) {
      if ((numSegs[i] > 0) && (end1[i] <= NodeCount)
          && (end2[i] <= NodeCount)) {
        index1 = end1[i] - 1;
        index2 = end2[i] - 1;
        xLength = X[index2] - X[index1];
        yLength = Y[index2] - Y[index1];
        zLength = Z[index2] - Z[index1];
        (*length)[i] = sqrt (xLength*xLength + yLength*yLength +
                             zLength*zLength) / numSegs[i];
      } else
        (*length)[i] = 0;
    }
  }
} /* end calculateSegmentLengths */

/****************************************************************************
 * Calculate segment lengths for tapered wires
 ****************************************************************************/

static void taperWireSegmentLengths ()
{
  int i, j;

  taperSeg1 = (float *) XtRealloc ((char *) taperSeg1,
                                   sizeof (float) * TaperWireCount);
  taperSeg2 = (float *) XtRealloc ((char *) taperSeg2,
                                   sizeof (float) * TaperWireCount);
  tLengthRad = (float *) XtRealloc ((char *) tLengthRad,
                                    sizeof (float) * TaperWireCount);
  for (i = 0; i < TaperWireCount; i++) {
    if (tSegLength[i] == 0) {
      taperSeg1[i] = 0;
      taperSeg2[i] = 0;
    } else {
      float length, delta, taperSeg;

      length = tSegLength[i] * GC_NS[i];

      /* RDEL is the ratio of the length of segment i+1 to i.  Thus, if
       * RDEL is one, then all segment lengths are equal.
       */
      if (GC_RDEL[i] == 1) {
        taperSeg1[i] = tSegLength[i];
        taperSeg2[i] = tSegLength[i];

      } else if (GC_IX[i] == 0) {
        delta = (1 - GC_RDEL[i]) / (1 - pow (GC_RDEL[i], GC_NS[i]));
        taperSeg = length * delta;
        taperSeg1[i] = taperSeg;
        if (GC_NS[i] > 1) {
          for (j = 0; j < GC_NS[i]; j++)
            taperSeg *= GC_RDEL[i];
        }
        taperSeg2[i] = taperSeg;

      /* DEL1 specifies the length of the 1st segment */
      } else if (GC_IX[i] == 1) {

        if ((GC_RDEL[i] >= length) || (GC_RDEL[i] < 0) || (GC_NS[i] == 1)) {
          taperSeg1[i] = 0;
          taperSeg2[i] = 0;
        } else {
          float del, dnewto, dnewt1, rmax, rd, dnewt, ornr, dn, rdx;
          int ns, nxstop;

          del = GC_RDEL[i];
          ns = GC_NS[i];
          dnewto = 2 * (ns*del-length)/(del*ns*(ns-1));
          dnewt1 = (4*length*(2-ns)*del*ns*(ns-5))/(3*del*ns*(1-ns));
          rmax = pow (length/del, 1/(ns-1));
          rd = 1;
          nxstop = 0;
          for (j = 0; j < 200; j++) {
            if (rd > rmax) rd = rmax;
            dn = pow (rd, ns);
            if (abs (dn-1) > 0.1) {
```

```
            omr = 1 - rd;
            dnewt = - omr*(length*omr-del*(1-dn))/(del*(1-dn-dn*ns*omr/rd));
           } else
            dnewt = dnewto + dnewt1*(rd-1);
           rd = rd -dnewt;
           if (nxstop == 1) break;
           if (abs (dnewt/rd) < 1.e-5) nxstop = 1;
          }
          rdx = rd;
          taperSeg = del;
          taperSeg1[i] = taperSeg;
          for (j = 0; j < GC_NS[i]; j++) taperSeg *= rdx;
          taperSeg2[i] = taperSeg;
         }
        }
       }
      }
      /* Determine the longest segment length and the smallest segment
       * length to radius ratio.
       */
      for (j = 0; j < TaperWireCount; j++) {
       float ratio1, ratio2;
       tSegLength[j] = taperSeg1[j] > taperSeg2[j] ? taperSeg1[j] :
        taperSeg2[j];
       ratio1 = taperSeg1[j] / GC_RAD1[j];
       ratio2 = taperSeg2[j] / GC_RAD2[j];
       tLengthRad[j] = ratio1 < ratio2 ? ratio1 : ratio2;
      }
     } /* end taperWireSegmentLengths */

     /*****************************************************************
      * Calculate segment lengths of Catenary wires
      *****************************************************************/

     static void catenaryWireSegmentLengths ()
     {
      int i;

      for (i = 0; i < CWireCount; i++) {
       if (cSegLength [i] > 0) {
        float icat;

        icat = CW_ICAT[i];
        if (icat == 3)
         cSegLength[i] = CW_RHM[i]/CW_NS[i];
        else {
         float xd, yd, zd, rhd, zhgt, length, c1, rh, ex2, exrp,
              exrm, exrs;
         int index1, index2;

         index1 = CW_END1[i] - 1;
         index2 = CW_END2[i] - 1;
         xd = X[index2] - X[index1];
         yd = Y[index2] - Y[index1];
         zd = Z[index2] - Z[index1];
         rhd = sqrt (xd*xd + yd*yd);
         xd = xd / rhd;
         yd = yd / rhd;
         if (icat == 1)
          zhgt = CW_ZM[i] - Z[index1];
         else if (icat == 2)
          zhgt = zd * CW_RHM[i] / rhd - CW_ZM[i];
         catsol (rhd, zd, CW_RHM[i], zhgt, &c1, &rh);
         if ((rh != 0) && (c1 != 0)) {
          catexp (rhd, rh, &ex2, &exrp, &exrm, &exrs);
          length = .5 * (exrp*c1 - exrm/c1);
         } else
          length = 0;
         cSegLength[i] = length / CW_NS[i];
        }
       }
      }
     } /* end catenaryWireSegmentLengths */

     /*****************************************************************
      * Calculate segment lengths of Wire Arcs
      *****************************************************************/

     static void wireArcSegmentLengths ()
     {
      int i;
      float angle1, angle2, arcLength;

      wSegLength = (float *) XtRealloc ((char *) wSegLength,
                        sizeof (float) * WireArcCount);
      for (i = 0; i < WireArcCount; i++) {

       /* Convert angles from degrees to radians */
       angle1 = GA_ANG1[i] * M_PI / 180;
       angle2 = GA_ANG2[i] * M_PI / 180;
       arcLength = (angle2 - angle1) * GA_RADA[i];
       wSegLength[i] = arcLength / GA_NS[i];
      }
```

```
} /* end wireArcSegmentLengths */

/*****************************************************************
 * Calculate segment lengths of Helix or Spiral Wires
 *****************************************************************/

static void helixSpiralSegmentLengths ()
{
  int i, j;
  float radrat, thmax, ahlx, ismall, hfac, x1, y1, z1, x2, y2, z2,
        sum, tinc, thet, hrad, zhlx;

  /* Allocate memory for the segment lengths */
  hSegLength = (float *) XtRealloc ((char *) hSegLength,
                                    sizeof (float) * HelixOrSpiralCount);
  hLengthRad = (float *) XtRealloc ((char *) hLengthRad,
                                    sizeof (float) * HelixOrSpiralCount);

  for (i = 0; i < HelixOrSpiralCount; i++) {

    radrat = pow ((GH_WR2[i] / GH_WR1[i]), 1 / (GH_NS[i] - 1));
    thmax = 2 * M_PI * fabs ((double) GH_TURNS[i]);
    if (GH_ISPX[i] == 0) { /* Log Spiral */
      ahlx = pow ((GH_HR2[i] / GH_HR1[i]), 1 / thmax);
      if (fabs (ahlx - 1) > 0.02) {
        ismall = 0;
        hfac = GH_ZLEN[i] / (GH_HR2[i] / GH_HR1[i] - 1);
      } else
        ismall = 1;
    } else
      ahlx = (GH_HR2[i] - GH_HR1[i]) / thmax;

    sum = 0;
    tinc = thmax / GH_NS[i];
    thet = 0;
    for (j = 0; j < GH_NS[i]; j++) {
      thet += tinc;
      if (j == 0) {
        x1 = GH_HR1[i];
        y1 = 0;
        z1 = 0;
      } else {
        x1 = x2;
        y1 = y2;
        z1 = z2;
      }
      if (j == GH_NS[i] - 1) {
        hrad = GH_HR2[i];
        zhlx = GH_ZLEN[i];
      } else {

        /* Log spiral */
        if (GH_ISPX[i] == 0) {
          hrad = GH_HR1[i] * pow (ahlx, thet);
          if (ismall == 0)
            zhlx = hfac * pow (ahlx, thet - 1);
          else
            zhlx = GH_ZLEN[i] * (thet / thmax) * (1 + .5 * (ahlx - 1) *
                                                  (thet - thmax));
        /* Archimedes spiral */
        } else {
          hrad = GH_HR1[i] + ahlx * thet;
          zhlx = GH_ZLEN[i] * thet / thmax;
        }
      }
      x2 = hrad * cos (thet);
      y2 = hrad * sin (thet);
      if (GH_TURNS[i] < 0) y2 = - y2;
      z2 = zhlx;
      sum += sqrt (pow (x2 - x1, 2) + pow (y2 - y1, 2) + pow (z2 - z1, 2));
    }
    hSegLength[i] = sum / GH_NS[i];
    hLengthRad[i] = GH_WR1[i] > GH_WR2[i] ? hSegLength[i] / GH_WR1[i] :
      hSegLength[i] / GH_WR2[i];
  }
} /* end helixSpiralSegmentLengths */

/*****************************************************************
 * Find wires which violate specified guidelines
 *****************************************************************/

/*****************************************************************
 * Find wires which violate segment length to wavelength ratio
 * guidelines
 *****************************************************************/

static void checkSegLengthToWavelength (segWaveError, segWaveWarning,
                                        segLength, count, wireType)
  float segWaveError, segWaveWarning, *segLength;
  int count;
  char *wireType;
{
  float segWave;
```

```
  int i;
  char string [80];
  extern float  DimensionsScale [];
  Boolean header = True;
  Boolean saveError;

  /* Record the errors if wire is straight or tapered */
  if (!strcmp (wireType, "Straight"))  errors = wireErrors;
  if (!strcmp (wireType, "Straight") || !strcmp (wireType, "Tapered"))
    saveError = True;
  else
    saveError = False;

  for (i = 0; i < count; i++) {
    segWave = segLength[i] * (DimensionsScale[DimIndex] / lowWavelength);
    if (segWave > segWaveError) {
      if (saveError)
        *errors |= SegLen2WaveLenError;
      /* Show header if not yet shown */
      if (header) {
        sprintf (string, "\nSegment length to wavelength:\n");
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
        header = False;
      }
      sprintf (string, "Error - %s %d (%g)\n", wireType, i+1, segWave);
      XmTextInsert (diagText, lastPosition, string);
      lastPosition += strlen (string);
    } else if (segWave > segWaveWarning) {
      if (saveError)
        *errors |= SegLen2WaveLenWarning;
      /* Show header if not yet shown */
      if (header) {
        sprintf (string, "\nSegment length to wavelength:\n");
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
        header = False;
      }
      sprintf (string, "Warning - %s %d (%g)\n", wireType, i+1, segWave);
      XmTextInsert (diagText, lastPosition, string);
      lastPosition += strlen (string);
    }
    if (saveError) errors++;
  }
} /* end checkSegLengthToWavelength */

/*****************************************************************************
 * Find wires which violate segment length to radius ratio
 * guidelines
 *****************************************************************************/

static void checkSegLengthToRadius (segRadError, segRadWarning,
                                    segLength, count, wireType, radius)
  float segRadError, segRadWarning, *segLength, *radius;
  int count;
  char *wireType;
{
  float segRad;
  int i;
  char string [132];
  Boolean header = True;
  Boolean saveError;

  /* Record the errors if wire is straight or tapered */
  if (!strcmp (wireType, "Straight"))  errors = wireErrors;
  if (!strcmp (wireType, "Straight") || !strcmp (wireType, "Tapered"))
    saveError = True;
  else
    saveError = False;

  for (i = 0; i < count; i++) {
    if (radius == NULL)
      segRad = segLength[i];
    else
      segRad = segLength[i] / radius[i];
    if (segRad < segRadError) {
      if (saveError)
        *errors |= SegLen2RadiusError;
      /* Show header if not yet shown */
      if (header) {
        sprintf (string, "\nSegment length to radius:\n");
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
        header = False;
      }
      sprintf (string, "Error - %s %d (%g)\n", wireType, i+1, segRad);
      XmTextInsert (diagText, lastPosition, string);
      lastPosition += strlen (string);
    } else if (segRad < segRadWarning) {
      if (saveError)
        *errors |= SegLen2RadiusWarning;
      /* Show header if not yet shown */
      if (header) {
```

```
        sprintf (string, "\nSegment length to radius:\n");
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
        header = False;
       }
       sprintf (string, "Warning - %s %d (%g)\n", wireType, i+1, segRad);
       XmTextInsert (diagText, lastPosition, string);
       lastPosition += strlen (string);
     }
     if (saveError) errors++;
   }
 } /* end checkSegLengthToRadius */

 /*****************************************************************************
  * Find wires which violate radius to wavelength guidelines
  *****************************************************************************/

 static void checkRadiusToWavelength (radWaveError, radWaveWarning,
                                      radius1, radius2, count, wireType)
  float radWaveError, radWaveWarning, *radius1, *radius2;
  int count;
  char *wireType;
 {
  float radius, radWave;
  int i;
  char string [132];
  Boolean header = True;
  Boolean saveError;

  /* Record the errors if wire is straight or tapered */
  if (!strcmp (wireType, "Straight"))  errors = wireErrors;
  if (!strcmp (wireType, "Straight") || !strcmp (wireType, "Tapered"))
    saveError = True;
  else
    saveError = False;

  for (i = 0; i < count; i++) {
    if (radius2 == NULL)
      radius = radius1[i];
    else
      radius = radius1[i] > radius2[i] ? radius1[i] : radius2[i];
    radWave = radius / lowWavelength;
    if (radWave > radWaveError) {
      if (saveError)
        *errors |= Radius2WaveLenError;
      /* Show header if not yet shown */
      if (header) {
        sprintf (string, "\nRadius to wavelength:\n");
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
        header = False;
      }
      sprintf (string, "Error - %s %d (%g)\n", wireType, i+1, radWave);
      XmTextInsert (diagText, lastPosition, string);
      lastPosition += strlen (string);
    } else if (radWave > radWaveWarning) {
      if (saveError)
        *errors |= Radius2WaveLenWarning;
      /* Show header if not yet shown */
      if (header) {
        sprintf (string, "\nRadius to wavelength:\n");
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
        header = False;
      }
      sprintf (string, "Warning - %s %d (%g)\n", wireType, i+1, radWave);
      XmTextInsert (diagText, lastPosition, string);
      lastPosition += strlen (string);
    }
    if (saveError) errors++;
  }
 } /* end checkRadiusToWavelength */

 /*****************************************************************************
  * Find wires which violate junction segment length ratios guidelines
  *****************************************************************************/

 static void checkSegLengthRatios (minSegs)
     float **minSegs;
 {
  enum WireTypes minType, maxType;
  int i, j, minIndex, maxIndex;
  float ratio, ratioError, ratioWarning, minLength, maxLength;
  Boolean header = True;
  char *text, string[132];

  text = XmTextGetString (diagWsegLengthRatio);
  ratioWarning = atof (text);
  XtFree (text);
  text = XmTextGetString (diagEsegLengthRatio);
  ratioError = atof (text);
  XtFree (text);
```

```
for (i = 0; i < NodeCount; i++) {
 minLength = 1.0e+38;
 maxLength = 0;

 /* Check straight wires for ends which use node i */
 for (j = 0; j < SWireCount; j++) {
  if ((GW_END1[j] == i+1) || (GW_END2[j] == i+1)) {
   if (sSegLength[j] < minLength) {
    minLength = sSegLength[j];
    minIndex = j;
    minType = Straight;
   }
   if (sSegLength[j] > maxLength) {
    maxLength = sSegLength[j];
    maxIndex = j;
    maxType = Straight;
   }
  }
 }

 /* Check Tapered Wires for ends which use node i */
 for (j = 0; j < TaperWireCount; j++) {
  if (GC_END1[j] == i+1) {
   if (taperSeg1[j] < minLength) {
    minLength = taperSeg1[j];
    minIndex = j;
    minType = Tapered;
   }
   if (taperSeg1[j] > maxLength) {
    maxLength = taperSeg1[j];
    maxIndex = j;
    maxType = Tapered;
   }
  } else if (GC_END2[j] == i+1) {
   if (taperSeg2[j] < minLength) {
    minLength = taperSeg2[j];
    minIndex = j;
    minType = Tapered;
   }
   if (taperSeg2[j] > maxLength) {
    maxLength = taperSeg2[j];
    maxIndex = j;
    maxType = Tapered;
   }
  }
 }

 /* Check Catenary Wires for ends which use node i */
 for (j = 0; j < CWireCount; j++) {
  if ((CW_END1[j] == i+1) || (CW_END2[j] == i+1)) {
   if (cSegLength[j] < minLength) {
    minLength = cSegLength[j];
    minIndex = j;
    minType = Catenary;
   }
   if (cSegLength[j] > maxLength) {
    maxLength = cSegLength[j];
    maxIndex = j;
    maxType = Catenary;
   }
  }
 }

 /* Save smallest segment info */
 minSegs[i][0] = minLength;
 minSegs[i][1] = (float) minType;
 minSegs[i][2] = (float) minIndex;

 /* Compute ratio and check for warning/errors */
 if (minLength > 0)
  ratio = maxLength / minLength;
 else
  ratio = 0;
 if (ratio > ratioError) {
  if (header) {
   sprintf (string, "\nJunction segment length ratios:\n");
   XmTextInsert (diagText, lastPosition, string);
   lastPosition += strlen (string);
   header = False;
  }
  if (minType == Straight) wireErrors[minIndex] |= JunctionSegLenRatioError;
  if (minType == Tapered) wireErrors[SWireCount + minIndex] |= JunctionSegLenRatioError;
  if (maxType == Straight) wireErrors[maxIndex] |= JunctionSegLenRatioError;
  if (maxType == Tapered) wireErrors[SWireCount + maxIndex] |= JunctionSegLenRatioError;
  sprintf (string, "Error - node %d %s %d (%g) %s %d (%g)\n",
        i+1, wireNames[minType], minIndex+1, minLength / lowWavelength,
        wireNames[maxType], maxIndex+1, maxLength / lowWavelength);
  XmTextInsert (diagText, lastPosition, string);
  lastPosition += strlen (string);
 } else if (ratio > ratioWarning) {
  if (header) {
   sprintf (string, "\nJunction segment length ratios:\n");
   XmTextInsert (diagText, lastPosition, string);
```

```
      lastPosition += strlen (string);
      header = False;
    }
    if (minType == Straight) wireErrors[minIndex] |= JunctionSegLenRatioWarning;
    if (minType == Tapered) wireErrors[SWireCount + minIndex] |= JunctionSegLenRatioWarning;
    if (maxType == Straight) wireErrors[maxIndex] |= JunctionSegLenRatioWarning;
    if (maxType == Tapered) wireErrors[SWireCount + maxIndex] |= JunctionSegLenRatioWarning;
    sprintf (string, "Warning - node %d %s %d (%g), %s %d (%g)\n",
          i+1, wireNames[minType], minIndex+1, minLength / lowWavelength,
          wireNames[maxType], maxIndex+1, maxLength / lowWavelength);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);
   }
   if (maxLength == 0) {
    sprintf (string, "node %d - no wires connected\n", i);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);
   }
  }
} /* end checkSegLengthRatios */

/*****************************************************************
 * Find wires which violate junction radius ratios guidelines
 *****************************************************************/

static void checkRadiusRatios (maxRadii)
    float **maxRadii;
{
 enum WireTypes minType, maxType;
 int i, j, minIndex, maxIndex;
 float ratio, ratioError, ratioWarning, minRadius, maxRadius;
 Boolean header = True;
 char *text, string[132];

 text = XmTextGetString (diagWradiusRatio);
 ratioWarning = atof (text);
 XtFree (text);
 text = XmTextGetString (diagEradiusRatio);
 ratioError = atof (text);
 XtFree (text);

 for (i = 0; i < NodeCount; i++) {
  minRadius = 1.0e+38;
  maxRadius = 0;

  /* Check straight wires for ends which use node i */
  for (j = 0; j < SWireCount; j++) {
   if ((GW_END1[j] == i+1) || (GW_END2[j] == i+1)) {
    if (GW_RAD[j] < minRadius) {
     minRadius = GW_RAD[j];
     minIndex = j;
     minType = Straight;
    }
    if (GW_RAD[j] > maxRadius) {
     maxRadius = GW_RAD[j];
     maxIndex = j;
     maxType = Straight;
    }
   }
  }

  /* Check Tapered Wires for ends which use node i */
  for (j = 0; j < TaperWireCount; j++) {
   if (GC_END1[j] == i+1) {
    if (GC_RAD1[j] < minRadius) {
     minRadius = GC_RAD1[j];
     minIndex = j;
     minType = Tapered;
    }
    if (GC_RAD1[j] > maxRadius) {
     maxRadius = GC_RAD1[j];
     maxIndex = j;
     maxType = Tapered;
    }
   } else if (GC_END2[j] == i+1) {
    if (GC_RAD2[j] < minRadius) {
     minRadius = GC_RAD2[j];
     minIndex = j;
     minType = Tapered;
    }
    if (GC_RAD2[j] > maxRadius) {
     maxRadius = GC_RAD2[j];
     maxIndex = j;
     maxType = Tapered;
    }
   }
  }

  /* Check Catenary Wires for ends which use node i */
  for (j = 0; j < CWireCount; j++) {
   if ((CW_END1[j] == i+1) || (CW_END2[j] == i+1)) {
    if (CW_RAD[j] < minRadius) {
     minRadius = CW_RAD[j];
```

```
        minIndex = j;
        minType = Catenary;
      }
      if (CW_RAD[j] > maxRadius) {
        maxRadius = CW_RAD[j];
        maxIndex = j;
        maxType = Catenary;
      }
    }
  }

  /* Save largest radius info */
  maxRadii[i][0] = maxRadius;
  maxRadii[i][1] = (float) maxType;
  maxRadii[i][2] = (float) maxIndex;

  /* Compute ratio and check for warning/errors */
  if (minRadius > 0)
    ratio = maxRadius / minRadius;
  else
    ratio = 0;
  if (ratio > ratioError) {
    if (header) {
      sprintf (string, "\nJunction radius ratios:\n");
      XmTextInsert (diagText, lastPosition, string);
      lastPosition += strlen (string);
      header = False;
    }
    if (minType == Straight) wireErrors[minIndex] |= JunctionRadiusRatioError;
    if (minType == Tapered) wireErrors[SWireCount + minIndex] |= JunctionRadiusRatioError;
    if (maxType == Straight) wireErrors[maxIndex] |= JunctionRadiusRatioError;
    if (maxType == Tapered) wireErrors[SWireCount + maxIndex] |= JunctionRadiusRatioError;
    sprintf (string, "Error - node %d %s %d (%g) %s %d (%g)\n",
          i+1, wireNames[minType], minIndex+1, minRadius / lowWavelength,
          wireNames[maxType], maxIndex+1, maxRadius / lowWavelength);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);
  } else if (ratio > ratioWarning) {
    if (header) {
      sprintf (string, "\nJunction radius ratios:\n");
      XmTextInsert (diagText, lastPosition, string);
      lastPosition += strlen (string);
      header = False;
    }
    if (minType == Straight) wireErrors[minIndex] |= JunctionRadiusRatioWarning;
    if (minType == Tapered) wireErrors[SWireCount + minIndex] |= JunctionRadiusRatioWarning;
    if (maxType == Straight) wireErrors[maxIndex] |= JunctionRadiusRatioWarning;
    if (maxType == Tapered) wireErrors[SWireCount + maxIndex] |= JunctionRadiusRatioWarning;
    sprintf (string, "Warning - node %d %s %d (%g), %s %d (%g)\n",
          i+1, wireNames[minType], minIndex+1, minRadius / lowWavelength,
          wireNames[maxType], maxIndex+1, maxRadius / lowWavelength);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);
  }
  if (maxRadius == 0) {
    sprintf (string, "No wires connected - node %d\n", i);
    XmTextInsert (diagText, lastPosition, string);
    lastPosition += strlen (string);
  }
 }
} /* end checkRadiusRatios */

/*****************************************************************************
 * Find match point errors.  These are found by determining the largest
 * radius and smallest segment at each node.  If the ratio of the
 * smallest segment to largest radius is less than or equal to 2.0
 * then it is a match point error.
 *****************************************************************************/

static void findMatchPointErrors (maxRadii, minSegs)
    float **maxRadii, **minSegs;
{
 int i, radiusType, segType;
 float ratio;
 Boolean header = True;
 char string [81];

 for (i = 0; i < NodeCount; i++) {
  /* Compute the ratio */
  if (maxRadii[i][0] > 0)
    ratio = minSegs[i][0] / maxRadii[i][0];
  else
    ratio = 0;

  /* Make sure that the largest radius & smallest segment are on
   * different wires.
   */
  if (maxRadii[i][2] != minSegs[i][2]) {
    if (ratio <= 2.0) {
      if (header) {
        sprintf (string, "\nMatch point errors:\n");
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
```

```
          header = False;
        }
        radiusType = (int) maxRadii[i][1];
        segType = (int) minSegs[i][1];
        if (radiusType == Straight)
          wireErrors[(int) maxRadii[i][2]] |= JunctionMatchPointError;
        if (radiusType == Tapered)
          wireErrors[SWireCount + (int) maxRadii[i][2]] |= JunctionMatchPointError;
        if (segType == Straight)
          wireErrors[(int) minSegs[i][2]] |= JunctionMatchPointError;
        if (segType == Tapered)
          wireErrors[SWireCount + (int) minSegs[i][2]] |= JunctionMatchPointError;
        sprintf (string, "%s %d %s %d\n",
              wireNames[radiusType], (int) maxRadii[i][2] + 1,
              wireNames[segType], (int) minSegs[i][2] + 1);
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
      }
    }
  }
} /* end findMatchPointErrors */

/*****************************************************************************/
static void catexp (x, rh, exr, exrp, exrm, exrs)
  float x, rh, *exr, *exrp, *exrm, *exrs;
{
  float xr;

  xr = x * rh;
  *exr = exp (xr);
  if (abs (xr) > .1) {
    *exrp = ((*exr) - 1)/rh;
    *exrm = (1/(*exr) - 1)/rh;
    *exrs = ((*exrm) + (*exrp))/rh;
  } else {
    *exrp = x * ((((8.333333E-3 * xr + .04166667) * xr + .1666667) *
       xr + .5) * xr + 1);
    *exrm = x * ((((-8.333333E-3 * xr + .04166667) * xr - .1666667) *
       xr + .5) * xr - 1);
    *exrs = x * x * (1 + .08333333 * xr * xr);
  }
} /* end catexp */

/*****************************************************************************/

static void catsol (x2, y2, xmx, ymx, c1, rh)
  float x2, y2, xmx, ymx, *c1, *rh;
{
  float xm, ym, rhdif, sqfac, cp, rxrx, yf, dyf, exrx, exrp, exrm, exrs;
  int i;

  xm = xmx;
  ym = ymx;
  *rh = 1.0;
  rhdif = 1.0;
  for (i = 0; i < 51; i++) {
    catexp (x2, *rh, &exrx, &exrp, &exrm, &exrs);
    sqfac = sqrt(y2*y2 + exrs);
    if (x2 < 0) sqfac = - sqfac;
    *c1 = (y2 + sqfac)/exrp;
    if (abs(rhdif/x2) < 1.E-5) break;
    if (i >= 51) {
      *rh = 0;
      *c1 = 0;
      break;
    }
    cp = (y2 + .5*(x2*(exrp-exrm)+2*y2*y2)/sqfac)/exrp - x2*exrx*(y2+sqfac)/
      (exrp*exrp);
    catexp (xm, *rh, &exrx, &exrp, &exrm, &exrs);
    rxrx = 1/exrx;
    yf = .5 * (exrp * (*c1) + exrm/(*c1));
    dyf = (-yf + .5 * (xm * (exrx * (*c1) - rxrx/(*c1))+ cp *
                (exrp-exrm/((*c1) * (*c1))))/(*rh);
    rhdif = (yf-ym)/dyf;
    *rh = (*rh) - rhdif;
    if ((*rh) * x2 > 50) *rh = 50 / x2;
  }
} /* end catsol */


static void findCrossedWires (void)
{
  char string [81], *text;
  int i, j, k, l, w1node1, w1node2, w2node1, w2node2,
     *w1end1, *w1end2, *w2end1, *w2end2;
  float dx2, dy2, dz2, radius1, radius2, minDistance, dx1, dy1, dz1,
      errorValue, crossx, crossy, crossz, v1, m11, m12, v2, m21, m22,
      mdet, vv, uu, xx, yy, zz, distance;
  Boolean header = True;

  /* Get error value */
  text = XmTextGetString (diagEwireRadii);
  errorValue = atof (text);
```

```
XtFree (text);

if ((SWireCount + TaperWireCount <= 0) || (errorValue <= 0))
 return;

for (i = 0; i < (SWireCount + TaperWireCount); i++) {

 if (i < SWireCount) {
  w2end1 = GW_END1;
  w2end2 = GW_END2;
  radius2 = GW_RAD[i];
  j = i;
 } else {
  w2end1 = GC_END1;
  w2end2 = GC_END2;
  j = i - SWireCount;
  radius2 = (GC_RAD1[j] + GC_RAD2[j]) / 2;
 }

 /* Compute incremental distances for wire two */
 w2node1 = w2end1[j] - 1;
 w2node2 = w2end2[j] - 1;
 dx2 = X[w2node2] - X[w2node1];
 dy2 = Y[w2node2] - Y[w2node1];
 dz2 = Z[w2node2] - Z[w2node1];

 for (k = i; k < (SWireCount + TaperWireCount); k++) {

  if (k < SWireCount) {
   w1end1 = GW_END1;
   w1end2 = GW_END2;
   radius1 = GW_RAD[k];
   l = k;
  } else {
   w1end1 = GC_END1;
   w1end2 = GC_END2;
   l = k - SWireCount;
   radius1 = (GC_RAD1[l] + GC_RAD2[l]) / 2;
  }

  /* Minimum distances interms of radii of two wires */
  minDistance = errorValue * (radius1 + radius2);

  /* Check for junctions */
  if ((w2end1[j] == w1end1[l]) || (w2end1[j] == w1end2[l]) ||
     (w2end2[j] == w1end1[l]) || (w2end2[j] == w1end2[l]))
   continue;

  /* Incremental distances for wire one */
  w1node1 = w1end1[l] - 1;
  w1node2 = w1end2[l] - 1;
  dx1 = X[w1node2] - X[w1node1];
  dy1 = Y[w1node2] - Y[w1node1];
  dz1 = Z[w1node2] - Z[w1node1];

  /* Cross terms for wire one and wire two */
  crossx = X[w1node1] - X[w2node1];
  crossy = Y[w1node1] - Y[w2node1];
  crossz = Z[w1node1] - Z[w2node1];

  /* Use partial derivatives of distance square.  Set-up the two
   * equations in two unknowns (uu, w).  Partial derivative with
   * respect to variable uu.
   */
  v1 = -(crossx * dx1 + crossy * dy1 + crossz * dz1);
  m11 = dx1 * dx1 + dy1 * dy1 + dz1 * dz1;
  m12 = -(dx2 * dx1 + dy2 * dy1 + dz2 * dz1);

  /* Partial derivative with respect to variable v */
  v2 = -(crossx * dx2 + crossy * dy2 + crossz * dz2);
  m21 = -m12;
  m22 = -(dx2 * dx2 + dy2 * dy2 + dz2 * dz2);

  /* Solve two equations in two unknows (uu, w) */
  mdet = m11 * m22 - m12 * m21;
  if (fabs(mdet) < 1.E-10)
   w = 0;
  else
   w = (v2 * m11 - m21 * v1) / mdet;

  /* Wire two is only defined between w = 0 to w = 1 */
  if (w < 0)
   w = 0;
  else if (w > 1)
   w = 1;
  uu = (v1 - m12 * w) / m11;

  /* Wire one is only defined between uu = 0 to uu = 1 */
  if (uu < 0) {
   uu = 0;
/*
   if ((m12 != 0)&&(w != 0)) w = v1/m12;
*/
```

```
      }
      else if (uu > 1) {
        uu = 1;
  /*
        if ((m12 != 0)&&(w != 0)) w = (v1 - m11) / m12;
  */
      }

      /* Find shortest distance between wires */
      xx = crossx + dx1 * uu - dx2 * w;
      yy = crossy + dy1 * uu - dy2 * w;
      zz = crossz + dz1 * uu - dz2 * w;
      distance = sqrt (xx * xx + yy * yy + zz * zz);
      if (minDistance > distance) {
        if (header) {
          sprintf (string, "\nCrossing wires:\n");
          XmTextInsert (diagText, lastPosition, string);
          lastPosition += strlen (string);
          header = False;
        }
        wireErrors[i] |= CrossedWireError;
        wireErrors[k] |= CrossedWireError;
        sprintf (string, "  wires %d and %d cross\n", i+1, k+1);
        XmTextInsert (diagText, lastPosition, string);
        lastPosition += strlen (string);
      }
    }
  }
} /* end findCrossedWires */


/*****************************************************************************
 * The following procedure runs both electrical & solution diagnostics.
 *
 * For every LOAD, VOLTAGE SOURCE, TRANSMISSION LINE, TWO PORT NETWORK,
 * and INSULATED WIRE, check that there is a legitimate wire associated it.
 *
 * In addition, check that the radius of the insulation is greater than the
 * radius of the insulated wire.
 *
 * For MAXIMUM COUPLING CALCULATION, check that the specified wires are
 * valid.  Determine if the wires specified for print of charges & currents
 * are valid.
 */
static void runElectricalDiagnostics ()
{
  Boolean checkPrintTag;
  int *tag1 [10], count1 [10], *tag2 [5], count2 [5], i, j, k, l;
  int printTag [1];
  float *radius1 [5], *radius2 [5];
  char string[81];
  char *label [10] = {
    "load", "voltage source", "transmission line", "transmission line",
    "two port network", "two port network", "insulated wire",
    "maximum coupling", "maximum coupling", "print option"};
  char *wireLabel [5] = {
    "straight wire", "tapered wire", "catenary wire", "wire arc",
    "helix or spiral wire"};

  /* Print title */
  XmTextSetString (diagText, "ELECTRICAL/SOLUTION DIAGNOSTICS\n\n");
  lastPosition = XmTextGetLastPosition (diagText);

  /* Initialize electrical arrays */
  tag1[0] = LD_Tag;
  tag1[1] = EX_Wire;
  tag1[2] = TL_Wire1;
  tag1[3] = TL_Wire2;
  tag1[4] = NT_Wire1;
  tag1[5] = NT_Wire2;
  tag1[6] = IS_ITAG;
  tag1[7] = CP_TAG1;
  tag1[8] = CP_TAG2;
  tag1[9] = printTag;
  if (PrintChargeCount)
    printTag[0] = PQ_IPTAQ[0];
  else
    printTag[0] = 0;

  count1[0] = LoadsCount;
  count1[1] = VoltageSourcesCount;
  count1[2] = TransmissionLinesCount;
  count1[3] = TransmissionLinesCount;
  count1[4] = TwoPortNetsCount;
  count1[5] = TwoPortNetsCount;
  count1[6] = InsulatedWiresCount;
  count1[7] = MaxCouplingCount;
  count1[8] = MaxCouplingCount;
  count1[9] = 1;

  /* Initialize Wire arrays */
  tag2[0] = GW_ITG;
  tag2[1] = GC_ITG;
```

```
tag2[2] = CW_ITG;
tag2[3] = GA_ITG;
tag2[4] = GH_ITG;

count2[0] = SWireCount;
count2[1] = TaperWireCount;
count2[2] = CWireCount;
count2[3] = WireArcCount;
count2[4] = HelixOrSpiralCount;

/* Initialize radius arrays */
radius1[0] = GW_RAD;
radius1[1] = GC_RAD1;
radius1[2] = CW_RAD;
radius1[3] = GA_RAD;
radius1[4] = GH_WR1;

radius2[0] = NULL;
radius2[1] = GC_RAD2;
radius2[2] = NULL;
radius2[3] = NULL;
radius2[4] = GH_WR2;

/* Determine whether Print Option tag needs to be checked */
if (PrintChargeCount && PrintCurrentCount)
checkPrintTag = !(PQ_IPTAQ[0] == 0) && (PQ_IPTFLQ[0] == 0 || PT_IPTFLQ[0] >= 0);
else
checkPrintTag = False;

for (i = 0; i < 10; i++) {
 int *tagArray = tag1[i];
 Boolean checkRadius = (i == 6) ? TRUE : FALSE;

 if (i == 9 && !checkPrintTag) continue;

 for (j = 0; j < count1[i]; j++) {
  float sheathRadius;
  Boolean found = FALSE;
  int currentTag = tagArray[j];

  if (checkRadius) sheathRadius = IS_RADI[j];

  /* For each wire type... */
  for (k = 0; k < 5; k++) {
   int *wireTag = tag2[k];

   /* Find a wire whose tag values matches 'currentTag' */
   for (l = 0; l < count2[k]; l++) {
    if (wireTag[l] == currentTag) {
     found = TRUE;

     /* If electrical input is insulated wire then make sure
      * that the radius of insulation is greater than the radius
      * of the insulated wire.*/
     if (checkRadius) {
      float rad, *rad1 = radius1[k], *rad2 = radius2[k];

      if (rad2 == NULL)
       rad = rad1[l];
      else
       rad = rad1[l] > rad2[l] ? rad1[l] : rad2[l];

      if (rad >= sheathRadius) {
       if (k == Straight) wireErrors[l] |= InvalidSheathRadiusError;
       if (k == Tapered) wireErrors[SWireCount + l] |= InvalidSheathRadiusError;
       sprintf
        (string,
         "INVALID insulated sheath radius specified for\n   %s %d\n",
         wireLabel[k], l+1);
       XmTextInsert (diagText, lastPosition, string);
       lastPosition += strlen (string);
      }
     }
    }
   }
  }

  /* Display error message if not found */
  if (!found) {
   sprintf (string, "INVALID wire specified for %s %d\n", label[i], j+1);
   XmTextInsert (diagText, lastPosition, string);
   lastPosition += strlen (string);
  }
 }
}
} /* end verifyTags */
```

fDiagnostics.c:

```
/*
 * fDiagnostics.c
 *
 * Procedures for creating the Diagnostics window
 */

#include "control.h"
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/PanedW.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>
#include "actionArea.h"

extern Widget topLevel;

/* Diagnostics Window Widgets */
Widget diagnosticsShell = NULL,
    diagIndividualWires,    /* Check boxes for Geometry */
    diagWireJunctions,      /*  Diagnostic options */
    diagCrossedWires,
    diagIndividualPatches,
    diagPatchWireJunctions,
    diagWsegmentLength,     /* Text boxes for geometry */
    diagWsegRadiusRatio1,   /*  guidelines */
    diagWradius,
    diagWsegLengthRatio,
    diagWradiusRatio,
    diagWedgeLength,
    diagWedgeSegRatio,
    diagWedgeRadiusRatio,
    diagEsegmentLength,
    diagEsegRadiusRatio1,
    diagEradius,
    diagEsegLengthRatio,
    diagEradiusRatio,
    diagEedgeLength,
    diagEwireRadii,
    diagEedgeSegRatio,
    diagEedgeRadiusRatio,
    diagGeometry,           /* Radio buttons for Diagnostic */
    diagElectrical,         /*  type */
    diagText;               /* Text widget for diagnostics */

extern void diagDestroyCB ();
extern void diagRunButtonCB ();
extern void diagPrintButtonCB ();
extern void diagVisualizeButtonCB ();
static void cancelButtonCB ();
static void createDiagnosticsWindow ();
static Widget createInputFields ();
static void createRunOptions ();

/*****************************************************************************/

void openDiagnosticsWindow ()
{
  if (diagnosticsShell == NULL) createDiagnosticsWindow ();

  /* Clear the scrollable text widget */
  XmTextSetString (diagText, "");

  XtPopup (diagnosticsShell, XtGrabNone);
}

/*****************************************************************************/

static void createDiagnosticsWindow ()
{
  Widget form, pane, w;
  Arg args [12];
  int n = 0;
  Position x, y;
  static ActionAreaItem actionItems[] = {
     {"Run",     diagRunButtonCB, NULL},
     {"Print",   diagPrintButtonCB, NULL},
                                        {"Visualize", diagVisualizeButtonCB, NULL},
     {"Close",   cancelButtonCB, NULL},
  };
  extern void newEscapeAction();

  XtTranslateCoords (topLevel, (Position) 0, (Position) 0, &x, &y);
  XtSetArg (args [n], XmNx, x); n++;
  XtSetArg (args [n], XmNy, y + 100); n++;
  diagnosticsShell =
  XtCreatePopupShell ("Diagnostics", topLevelShellWidgetClass,
                topLevel, args, n);

  XtAddCallback (diagnosticsShell, XmNdestroyCallback, diagDestroyCB, NULL);
```

```
    newEscapeAction(diagnosticsShell);

    form = XmCreateForm (diagnosticsShell, "form", NULL, 0);
    XtManageChild (form);

    XtSetArg (args[n], XmNsashWidth, 1); n++;
    XtSetArg (args[n], XmNsashHeight, 1); n++;
    XtSetArg (args[n], XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
    pane = XmCreatePanedWindow (form, "pane", args, n);
    XtManageChild (pane);

    /* Create text output */
    n = 0;
    XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg (args[n], XmNtopOffset, 5); n++;
    XtSetArg (args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
    XtSetArg (args[n], XmNleftWidget, pane); n++;
    XtSetArg (args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
    XtSetArg (args[n], XmNbottomWidget, pane); n++;
    XtSetArg (args[n], XmNbottomOffset, 5); n++;
    XtSetArg (args[n], XmNrightAttachment, XmATTACH_FORM); n++;
    XtSetArg (args[n], XmNrightOffset, 5); n++;
    XtSetArg (args[n], XmNeditMode, XmMULTI_LINE_EDIT); n++;
    XtSetArg (args[n], XmNeditable, FALSE); n++;
    XtSetArg (args[n], XmNcolumns, 50); n++;
    diagText = XmCreateScrolledText (form, "text", args, n);
    XtManageChild (diagText);

    form = XmCreateForm (pane, "form", NULL, 0);
    XtManageChild (form);

    n = 0;
    XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg (args [n], XmNbottomAttachment, XmATTACH_FORM); n++;
    form = XmCreateForm (form, "form2", args, n);
    XtManageChild (form);

    w = createInputFields (form);

    createRunOptions (pane);

    createActionArea (pane, actionItems, XtNumber (actionItems));

} /* end createDiagnosticWindows */

/*****************************************************************************
 * Creates input fields for geometry guidelines
 *****************************************************************************/

static Widget createInputFields (parent)
  Widget parent;
{
  XmString xmstring;
  Arg args [10];
  int n = 0;
  Widget label, rowColumn;

  xmstring = XmStringCreateLtoR ("MODELING GUIDELINES:",
                  XmSTRING_DEFAULT_CHARSET);
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNleftOffset, 10); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNtopOffset, 10); n++;
  XtSetArg (args [n], XmNlabelString, xmstring); n++;
  label = XmCreateLabel (parent, "headerLabel", args, n);
  XtManageChild (label);
  XmStringFree (xmstring);

  n = 0;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNleftOffset, 340); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNtopWidget, label); n++;
  XtSetArg (args [n], XmNtopOffset, 15); n++;
  label = XmCreateLabel (parent, "Warning", args, n);
  XtManageChild (label);

  n = 0;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNleftWidget, label); n++;
  XtSetArg (args [n], XmNleftOffset, 60); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNtopWidget, label); n++;
  label = XmCreateLabel (parent, "Error", args, n);
  XtManageChild (label);

  /* Create the row column box */
  n = 0;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
```

```
XtSetArg (args [n], XmNtopWidget, label); n++;
XtSetArg (args [n], XmNpacking, XmPACK_COLUMN); n++;
rowColumn = XmCreateRowColumn (parent, "rowColumn", args, n);
XtManageChild (rowColumn);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNvalue, ".2"); n++;
diagEsegmentLength = XmCreateText (rowColumn, "", args, n);
XtManageChild (diagEsegmentLength);

XtSetArg (args [2], XmNvalue, "2.0");
diagEsegRadiusRatio1 = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagEsegRadiusRatio1);

XtSetArg (args [2], XmNvalue, ".03");
diagEradius = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagEradius);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

XtSetArg (args [2], XmNvalue, "5.0");
diagEsegLengthRatio = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagEsegLengthRatio);

XtSetArg (args [2], XmNvalue, "100.0");
diagEradiusRatio = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagEradiusRatio);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

diagEwireRadii = XmCreateText (rowColumn, "text", args, n-1);
XtManageChild (diagEwireRadii);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

XtSetArg (args [2], XmNvalue, ".25");
XtSetArg (args [3], XmNsensitive, FALSE);
diagEedgeLength = XmCreateText (rowColumn, "text", args, 4);
XtManageChild (diagEedgeLength);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

XtSetArg (args[2], XmNsensitive, FALSE);
diagEedgeSegRatio = XmCreateText (rowColumn, "text", args, 3);
XtManageChild (diagEedgeSegRatio);

diagEedgeRadiusRatio = XmCreateText (rowColumn, "text", args, 3);
XtManageChild (diagEedgeRadiusRatio);

/* Create the row column box */
n = 0;
XtSetArg (args [n], XmNrightAttachment, XmATTACH_WIDGET); n++;
XtSetArg (args [n], XmNrightWidget, rowColumn); n++;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg (args [n], XmNtopWidget, rowColumn); n++;
XtSetArg (args [n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg (args [n], XmNbottomWidget, rowColumn); n++;
XtSetArg (args [n], XmNpacking, XmPACK_COLUMN); n++;
rowColumn = XmCreateRowColumn (parent, "rowColumn", args, n);
XtManageChild (rowColumn);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
XtSetArg (args [n], XmNcolumns, 11); n++;
XtSetArg (args [n], XmNvalue, ".1"); n++;
diagWsegmentLength = XmCreateText (rowColumn, "", args, n);
XtManageChild (diagWsegmentLength);

XtSetArg (args [2], XmNvalue, "2.0");
diagWsegRadiusRatio1 = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagWsegRadiusRatio1);

XtSetArg (args [2], XmNvalue, ".01");
diagWradius = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagWradius);
```

```
/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

XtSetArg (args [2], XmNvalue, "2.0");
diagWsegLengthRatio = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagWsegLengthRatio);

XtSetArg (args [2], XmNvalue, "10.0");
diagWradiusRatio = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagWradiusRatio);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

XtSetArg (args [2], XmNvalue, ".1");
XtSetArg (args [3], XmNsensitive, FALSE);
diagWedgeLength = XmCreateText (rowColumn, "text", args, 4);
XtManageChild (diagWedgeLength);

/* Dummy */
label = XmCreateLabel (rowColumn, "", NULL, 0);
XtManageChild (label);

XtSetArg (args[2], XmNsensitive, FALSE);
diagWedgeSegRatio = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagWedgeSegRatio);

diagWedgeRadiusRatio = XmCreateText (rowColumn, "text", args, n);
XtManageChild (diagWedgeRadiusRatio);

/* Create the row column box */
n = 0;
XtSetArg (args [n], XmNrightAttachment, XmATTACH_WIDGET); n++;
XtSetArg (args [n], XmNrightWidget, rowColumn); n++;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg (args [n], XmNtopWidget, rowColumn); n++;
XtSetArg (args [n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg (args [n], XmNbottomWidget, rowColumn); n++;
XtSetArg (args [n], XmNfractionBase, 14); n++;
rowColumn = XmCreateForm (parent, "rowColumn", args, n);
/*
XtSetArg (args [n], XmNpacking, XmPACK_COLUMN); n++;
rowColumn = XmCreateRowColumn (parent, "rowColumn", args, n);
*/
XtManageChild (rowColumn);

/* Widgets for Individual Wires input */
n = 0;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_POSITION); n++;
XtSetArg (args [n], XmNbottomAttachment, XmATTACH_POSITION); n++;
XtSetArg (args [n], XmNtopPosition, 0); n++;
XtSetArg (args [n], XmNbottomPosition, 1); n++;
diagIndividualWires = XmCreateToggleButton (rowColumn, "INDIVIDUAL WIRES",
                                args, n);
XtManageChild (diagIndividualWires);

n = 2;
XtSetArg (args [n], XmNtopPosition, 1); n++;
XtSetArg (args [n], XmNbottomPosition, 2); n++;
label = XmCreateLabel (rowColumn,
                " segment length (wavelengths)  >", args, n);
XtManageChild (label);

n = 2;
XtSetArg (args [n], XmNtopPosition, 2); n++;
XtSetArg (args [n], XmNbottomPosition, 3); n++;
label = XmCreateLabel (rowColumn,
                " segment/radius ratio          <", args, n);
XtManageChild (label);

n = 2;
XtSetArg (args [n], XmNtopPosition, 3); n++;
XtSetArg (args [n], XmNbottomPosition, 4); n++;
label = XmCreateLabel (rowColumn,
                " radius (wavelengths)          >", args, n);
XtManageChild (label);

n = 2;
XtSetArg (args [n], XmNtopPosition, 4); n++;
XtSetArg (args [n], XmNbottomPosition, 5); n++;
diagWireJunctions = XmCreateToggleButton (rowColumn,
                                "WIRE JUNCTIONS", args, n);
```

```
  XtManageChild (diagWireJunctions);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 5); n++;
  XtSetArg (args [n], XmNbottomPosition, 6); n++;
  label = XmCreateLabel (rowColumn,
                " segment length ratio        >", args, n);
  XtManageChild (label);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 6); n++;
  XtSetArg (args [n], XmNbottomPosition, 7); n++;
  label = XmCreateLabel (rowColumn,
                " radius ratio                >", args, n);
  XtManageChild (label);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 7); n++;
  XtSetArg (args [n], XmNbottomPosition, 8); n++;
  diagCrossedWires = XmCreateToggleButton
    (rowColumn, "CROSSED STRAIGHT WIRES", args, n);
  XtManageChild (diagCrossedWires);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 8); n++;
  XtSetArg (args [n], XmNbottomPosition, 9); n++;
  label = XmCreateLabel
    (rowColumn, " number of wire radii        =", args, n);
  XtManageChild (label);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 9); n++;
  XtSetArg (args [n], XmNbottomPosition, 10); n++;
  XtSetArg (args [n], XmNsensitive, FALSE); n++;
  diagIndividualPatches = XmCreateToggleButton
    (rowColumn, "INDIVIDUAL PATCHES", args, n);
  XtManageChild (diagIndividualPatches);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 10); n++;
  XtSetArg (args [n], XmNbottomPosition, 11); n++;
  XtSetArg (args [n], XmNsensitive, FALSE); n++;
  label = XmCreateLabel
    (rowColumn, " edge length (wavelengths)    >", args, n);
  XtManageChild (label);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 11); n++;
  XtSetArg (args [n], XmNbottomPosition, 12); n++;
  XtSetArg (args [n], XmNsensitive, FALSE); n++;
  diagPatchWireJunctions =
    XmCreateToggleButton (rowColumn, "PATCH/WIRE JUNCTIONS", args, n);
  XtManageChild (diagPatchWireJunctions);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 12); n++;
  XtSetArg (args [n], XmNbottomPosition, 13); n++;
  XtSetArg (args [n], XmNsensitive, FALSE); n++;
  label =
    XmCreateLabel (rowColumn, " edge/segment ratio        >", args, n);
  XtManageChild (label);

  n = 2;
  XtSetArg (args [n], XmNtopPosition, 13); n++;
  XtSetArg (args [n], XmNbottomPosition, 14); n++;
  XtSetArg (args [n], XmNsensitive, FALSE); n++;
  label =
    XmCreateLabel (rowColumn, " edge/radius ratio        >", args, n);
  XtManageChild (label);


  return (rowColumn);

} /* end createInputFields */

/*************************************************************************
 * Adds Run Options radio buttons to window
 *************************************************************************/

static void createRunOptions (parent)
  Widget parent;
{
  Arg args [10];
  int n = 0;
  Widget form, label, rowColumn;

  form = XmCreateForm (parent, "form", NULL, 0);
  XtManageChild (form);

  n = 0;
  XtSetArg (args [n], XmNmarginTop, 7); n++;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNleftOffset, 10); n++;
```

```
  label = XmCreateLabel (form, "OPTIONS:", args, n);
  XtManageChild (label);

  /* Create the row column box */
  n = 0;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNtopWidget, label); n++;
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNleftWidget, label); n++;
  XtSetArg (args [n], XmNleftOffset, 15); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNrightOffset, 10); n++;
  XtSetArg (args [n], XmNpacking, XmPACK_TIGHT); n++;
  XtSetArg (args [n], XmNorientation, XmHORIZONTAL); n++;
  XtSetArg (args [n], XmNspacing, 10); n++;
  rowColumn = XmCreateRadioBox (form, "radioBox", args, n);

  /* Create the Options buttons */
  XtSetArg (args[0], XmNset, TRUE);
  diagGeometry = XmCreateToggleButton (rowColumn, "Geometry", args, 1);
  XtManageChild (diagGeometry);
  diagElectrical = XmCreateToggleButton (rowColumn,
     "Electrical/Solution", NULL, 0);
  XtManageChild (diagElectrical);
  XtManageChild (rowColumn);

}

/*****************************************************************/

static void cancelButtonCB (void)
{
  XtPopdown (diagnosticsShell);

} /* end cancelButtonCB */
```

A.11 needsplt.c:

needsplt.c:

```
/*  needsplt.c   */
/*  modified by LCR 5/17/94 */

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/DrawingA.h>
#include <srgp.h>
#include "tplx11.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define       FULL_CIRCLE             (360*64)
#define START_CIRCLE   0
#define MAXLINE 120

extern Widget   topLevel;
extern void     createMessageDialog ();

static void     destroyCB ();
static void     drawing_linear();
static void     drawing_smith();
static void     drawing_polar();
static int      getZ();
static int      getA();
static int      getCR();
static int      getQ();
static int      getCP();
static int      getPT();
static int      getNE();
static int      getNH();
static XtEventHandler LinearEventHandler();
static XtEventHandler SmithEventHandler();
static XtEventHandler PolarEventHandler();

char          line[120];
char          file_name[132];
FILE          *infile;
char          in_file[132];
static char     rz[]  = ".rz";
static char     ra[]  = ".ra";
static char     rcr[] = ".rcr";
static char     rq[]  = ".rq";
static char     rcp[] = ".rcp";
static char     rpt[] = ".rpt";
static char     me[] = ".me";
static char     mm[] = ".mm";
int           plttype;
int           source;
float         selfreq;
float         selx,sely,selz;
float         angle;
int           tag;
FILE          *filep;
Boolean       sphigsOff = True;

/* Plotting data */

typedef struct _plotData {
  int           numxpts;
  float         xarray[2500],
                yarray[2500];
  float         freq[300];
  float         xlunit,
                xrunit,
                ytunit,
                ybunit;
  char          titlestr[80];
  GC            gc;
} PlotData;

#include <Xm/PushB.h>
#include <Xm/Form.h>
void needsPlot (type, inputs)
  char        *type,
              *inputs;     /* string containing input values */
{
  char        *ptr;
  Widget        popup, drawing_a;
  Arg         args[4];
  int         n,
              cont;
  PlotData      *plotData;
  extern Widget topLevel;
  extern char   *necOutputFilename;
```

```
extern void newEscapeAction2();

/* Create the plotting window */
popup = XtVaCreatePopupShell
  (NULL, topLevelShellWidgetClass, topLevel,
   XmNtitle, "Plot Output",
   XmNallowShellResize, True,
   XmNdeleteResponse, XmDESTROY,
   NULL);

newEscapeAction2(popup);

n = 0;
XtSetArg(args[n],XtNx,0); n++;
XtSetArg(args[n],XtNy,0); n++;
XtSetArg(args[n],XtNwidth,701); n++;
XtSetArg(args[n],XtNheight,701); n++;

drawing_a = XmCreateDrawingArea(popup,"window",args,n);

XtAddCallback (drawing_a, XmNdestroyCallback, destroyCB, NULL);

/* Create structure for storing plotting data. Save GC */
plotData = (PlotData *) XtMalloc (sizeof (PlotData));
XtVaSetValues (drawing_a, XmNuserData, plotData, NULL);

/* determine and open input files */
strcpy(file_name,necOutputFilename);
ptr = strrchr(file_name,'.');
if (ptr) *(ptr) = '\0';

/* get the command option */
if (strcmp(type,"Z") == 0) {  /* Impedance */
  sscanf (inputs, "%i %i", &plttype, &source);
  cont = getZ (drawing_a);
}
else if (strcmp(type,"A") == 0) {  /* Admittance */
  sscanf (inputs, "%i %i", &plttype, &source);
  cont = getA (drawing_a);
}
else if (strcmp(type,"CR") == 0) { /* Currents */
  sscanf (inputs, "%i %i %f %i", &plttype, &source, &selfreq, &tag);
  cont = getCR (drawing_a);
}
else if (strcmp(type,"Q") == 0) {  /* Charge */
  sscanf (inputs, "%i %i %f %i", &plttype, &source, &selfreq, &tag);
  cont = getQ (drawing_a);
}
else if (strcmp(type,"CP") == 0) {   /* Coupling */
  cont = getCP (drawing_a);
}
else if (strcmp(type,"PT") == 0) {   /* Radiation Patterns */
  sscanf (inputs, "%i %i %f %f", &plttype, &source, &selfreq, &angle);
  cont = getPT (drawing_a);
}
else if (strcmp(type,"NE") == 0) {   /* Near Electric Fields */
  float sel1, sel2, sel3;

  sscanf (inputs, "%i %i %f %f %f", &plttype, &source, &sel1, &sel2, &sel3);
  switch (plttype) {
    case 1:
    case 5:  sely = sel1;
             selz = sel2;
             selfreq = sel3;
             break;
    case 2:
    case 6:  selx = sel1;
             selz = sel2;
             selfreq = sel3;
             break;
    case 3:
    case 7:  selx = sel1;
             sely = sel2;
             selfreq = sel3;
             break;
    case 4:
    case 8:  selx = sel1;
             sely = sel2;
             selz = sel3;
             break;
  }
  cont = getNE (drawing_a);
}
else if (strcmp(type,"NH") == 0) {   /* Near Magnetic Fields */
  float sel1, sel2, sel3;

  sscanf (inputs, "%i %i %f %f %f", &plttype, &source, &sel1, &sel2, &sel3);
  switch (plttype) {
    case 1:
    case 5:  sely = sel1;
             selz = sel2;
             selfreq = sel3;
             break;
```

```
        case 2:
        case 6:  selx = sel1;
                 selz = sel2;
                 selfreq = sel3;
                 break;
        case 3:
        case 7:  selx = sel1;
                 sely = sel2;
                 selfreq = sel3;
                 break;
        case 4:
        case 8:  selx = sel1;
                 sely = sel2;
                 selz = sel3;
                 break;
      }
      cont = getNH (drawing_a);
    }

    /* If there were input errors, don't plot */
    if (cont == -1) {
       XtDestroyWidget (popup);
       XtFree ((char *) plotData);
       return;
    }

    /* Add callbacks */
    if ((strcmp (type,"Z") == 0) && (plttype == 3))
      XtAddEventHandler(drawing_a, ExposureMask, TRUE,
                (XtEventHandler) SmithEventHandler, 0);
    else if (strcmp (type,"PT") == 0)
      XtAddEventHandler(drawing_a, ExposureMask, TRUE,
                (XtEventHandler) PolarEventHandler, 0);
    else
      XtAddEventHandler(drawing_a, ExposureMask, TRUE,
                (XtEventHandler) LinearEventHandler, 0);
    XtManageChild (drawing_a);
    XtPopup (popup, XtGrabNone);

    SIMPLE_DISP = XtDisplay(drawing_a);
    SIMPLE_WIN = XtWindow(drawing_a);
    SIMPLE_WIN_WIDTH = 700;
    SIMPLE_WIN_HIGHT = 700;

    /* create SRGP window */
    if (sphigsOff) {
      sphigsOff = False;
    }
    SRGP_begin("needsplt", 700, 700, 3, FALSE);

    SRGP_loadCommonColor (0, "white");
    SRGP_loadCommonColor (1, "black");
    SRGP_loadCommonColor (2, "white");
    SRGP_loadCommonColor (4, "red");

    if ((strcmp (type,"Z") == 0) && (plttype == 3))
      drawing_smith(drawing_a);
    else if (strcmp (type,"PT") == 0)
      drawing_polar(drawing_a);
    else
      drawing_linear(drawing_a);

} /* end needsPlot */

static XtEventHandler SmithEventHandler(Widget widget, char *xtag, XEvent *xevent)
{
  SIMPLE_DISP = XtDisplay(widget);
  SIMPLE_WIN = XtWindow(widget);

  if (xevent->type==Expose)
    drawing_smith(widget);
  xtag = xtag;
  return (XtEventHandler)0;
}

static XtEventHandler PolarEventHandler(Widget widget, char *xtag, XEvent *xevent)
{
  SIMPLE_DISP = XtDisplay(widget);
  SIMPLE_WIN = XtWindow(widget);

  if (xevent->type==Expose)
    drawing_polar(widget);
  xtag = xtag;
  return (XtEventHandler)0;
}

static XtEventHandler LinearEventHandler(Widget widget, char *xtag, XEvent *xevent)
{
  SIMPLE_DISP = XtDisplay(widget);
  SIMPLE_WIN = XtWindow(widget);

  if (xevent->type==Expose)
    drawing_linear(widget);
```

```
  xtag = xtag;
  return (XtEventHandler)0;
}

static int getZ(drawing_a)
  Widget       drawing_a;
{
  static char *Zstr[3] = {
    "Resistance vs Frequency",
    "Reactance vs Frequency",
    "Smith Chart"};
  char     lookfor[15];
  int      i;
  float    a,b,c,d,e;
  PlotData  *plotData;
  extern FILE *efopen ();

  /* Get data structure from widget for storing plot data */
  XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);

  sprintf(plotData->titlestr,"%s, Source =%5d",Zstr[plttype-1],source);
  strcpy(in_file,file_name);
  strcat(in_file,rz);
  if ((infile = efopen (in_file, "r")) == NULL)
    return (0);

  /* dump first lines */
  sprintf(lookfor,"Source :%5d\n",source);
  fgets(line, MAXLINE, infile);
  while(strcmp(lookfor,line) != 0) {
    if (fgets(line, MAXLINE, infile) == NULL) {
      createMessageDialog (topLevel, "Impedance",
                  "Could not plot.  Source not found.", XmDIALOG_MESSAGE);
      return (-1);
    }
  }

  /* now read in data */
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);
  i = 1;
  while ((fscanf(infile,"%f %f %f %f %f",&a,&b,&c,&d,&e)) == 5) {
    if (plttype == 1) plotData->yarray[i] = b;
    if (plttype == 2) plotData->yarray[i] = c;
    plotData->xarray[i] = a;
    if (plttype == 3) {
      plotData->freq[i] = a;
      plotData->xarray[i] = d;
      plotData->yarray[i] = e;
    }
    if (i == 1) {
      plotData->xlunit = plotData->xarray[1];
      plotData->xrunit = plotData->xarray[1];
      plotData->ytunit = plotData->yarray[1];
      plotData->ybunit = plotData->yarray[1];
    } else {
      if (plotData->xarray[i] < plotData->xlunit)
        plotData->xlunit = plotData->xarray[i];
      if (plotData->xarray[i] > plotData->xrunit)
        plotData->xrunit = plotData->xarray[i];
      if (plotData->yarray[i] < plotData->ybunit)
        plotData->ybunit = plotData->yarray[i];
      if (plotData->yarray[i] > plotData->ytunit)
        plotData->ytunit = plotData->yarray[i];
    }
    i++;
  }
  fclose(infile);
  plotData->numxpts = i-1;
  if (i-1 == 0) {
    createMessageDialog (topLevel, "Impedance",
                  "Graph contains no data points.", XmDIALOG_MESSAGE);
    return (-1);
  } else
    return (0);
}

static int getA (drawing_a)
  Widget       drawing_a;
{
  static char *Astr[2] = {
    "Conductance vs Wavelength",
    "Susceptance vs Wavelength"};
  char     lookfor[15];
  int      i;
  float    a,b,c;
  PlotData  *plotData;
  extern FILE *efopen ();

  /* Get data structure from widget for storing plot data */
  XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);
```

```
  sprintf(plotData->titlestr,"%s, Source =%5d",Astr[plttype-1],source);
  strcpy(in_file,file_name);
  strcat(in_file,ra);
  if ((infile = efopen (in_file, "r")) == NULL)
    return (0);

  /* dump first lines */
  sprintf(lookfor,"Source :%5d\n",source);
  fgets(line, MAXLINE, infile);
  while(strcmp(lookfor,line) != 0) {
    if (fgets(line, MAXLINE, infile) == NULL) {
      createMessageDialog (topLevel, "Admittance",
                "Could not plot.  Source not found.", XmDIALOG_MESSAGE);
      return (-1);
    }
  }

  /* now read in data */
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);
  i = 1;
  while ((fscanf(infile,"%f %f %f",&a,&b,&c)) == 3) {
    if (plttype == 1) plotData->yarray[i] = b;
    if (plttype == 2) plotData->yarray[i] = c;
    plotData->xarray[i] = a;
    if (i == 1) {
      plotData->xlunit = plotData->xarray[1];
      plotData->xrunit = plotData->xarray[1];
      plotData->ytunit = plotData->yarray[1];
      plotData->ybunit = plotData->yarray[1];
    } else {
      if (plotData->xarray[i] < plotData->xlunit)
        plotData->xlunit = plotData->xarray[i];
      if (plotData->xarray[i] > plotData->xrunit)
        plotData->xrunit = plotData->xarray[i];
      if (plotData->yarray[i] < plotData->ybunit)
        plotData->ybunit = plotData->yarray[i];
      if (plotData->yarray[i] > plotData->ytunit)
        plotData->ytunit = plotData->yarray[i];
    }
    i++;
  }
  fclose(infile);
  plotData->numxpts = i-1;
  if (i-1 == 0) {
    createMessageDialog (topLevel, "Admittance",
      "Graph contains no data points.", XmDIALOG_MESSAGE);
    return (-1);
  } else
    return (0);
}

static int getCR (drawing_a)
  Widget        drawing_a;
{
  static char *CRstr[8] = {
    "Current Mag vs X",
    "Current Mag vs Y",
    "Current Mag vs Z",
    "Current Mag vs Seg",
    "Current Phase vs X",
    "Current Phase vs Y",
    "Current Phase vs Z",
    "Current Phase vs Seg"};
  int i;
  float a,b,c,d,e,f;
  int ia,ib,ic;
  PlotData *plotData;
  extern FILE *efopen ();

  /* Get data structure from widget for storing plot data */
  XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);

  if (tag == 0)
    sprintf(plotData->titlestr,"%s, Source =%5d, Freq =%6.2f, Tag = ALL",
      CRstr[plttype-1], source,selfreq);
  else
    sprintf(plotData->titlestr,"%s, Source =%5d, Freq =%6.2f, Tag =%3d",
      CRstr[plttype-1], source,selfreq,tag);
  strcpy(in_file,file_name);
  strcat(in_file,rcr);
  if ((infile = efopen (in_file, "r")) == NULL)
    return (0);

  /* dump first 6 lines */
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);

  /* now read in data */
```

```
 i = 1;
 while ((fscanf(infile,"%d %d %f %f %f %f %f %d",
       &ia,&ib,&a,&b,&c,&d,&e,&f,&ic)) != EOF) {
  if ((source == ic)&&(selfreq == f)&&((tag == ib)||(tag == 0))) {
   switch (plttype)
   {
   case 1: plotData->yarray[i] = d;
         plotData->xarray[i] = a;
         break;
   case 2: plotData->yarray[i] = d;
         plotData->xarray[i] = b;
         break;
   case 3: plotData->yarray[i] = d;
         plotData->xarray[i] = c;
         break;
   case 4: plotData->yarray[i] = d;
         plotData->xarray[i] = (float) ia;
         break;
   case 5: plotData->yarray[i] = e;
         plotData->xarray[i] = a;
         break;
   case 6: plotData->yarray[i] = e;
         plotData->xarray[i] = b;
         break;
   case 7: plotData->yarray[i] = e;
         plotData->xarray[i] = c;
         break;
   case 8: plotData->yarray[i] = e;
         plotData->xarray[i] = (float) ia;
         break;
   }
   if (i == 1) {
    plotData->xlunit = plotData->xarray[1];
    plotData->xrunit = plotData->xarray[1];
    plotData->ytunit = plotData->yarray[1];
    plotData->ybunit = plotData->yarray[1];
   } else {
    if (plotData->xarray[i] < plotData->xlunit)
     plotData->xlunit = plotData->xarray[i];
    if (plotData->xarray[i] > plotData->xrunit)
     plotData->xrunit = plotData->xarray[i];
    if (plotData->yarray[i] < plotData->ybunit)
     plotData->ybunit = plotData->yarray[i];
    if (plotData->yarray[i] > plotData->ytunit)
     plotData->ytunit = plotData->yarray[i];
   }
   i++;
  }
 }
 fclose(infile);
 plotData->numxpts = i-1;
 if (i-1 == 0) {
  createMessageDialog (topLevel, "Currents",
    "Graph contains no data points.", XmDIALOG_MESSAGE);
  return (-1);
 } else
  return (0);
}

static int getQ (drawing_a)
 Widget      drawing_a;
{
 static char *Qstr[4] = {
  "Charge vs X",
  "Charge vs Y",
  "Charge vs Z",
  "Charge vs Seg"};
 int i;
 float a,b,c,d,e;
 int ia,ib,ic;
 char ch;
 PlotData *plotData;
 extern FILE *efopen ();

 /* Get data structure from widget for storing plot data */
 XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);

 if (tag == 0)
  sprintf(plotData->titlestr,"%s, Source =%5d, Freq =%6.2f, Tag = ALL",
    Qstr[plttype-1], source,selfreq);
 else
  sprintf(plotData->titlestr,"%s, Source =%5d, Freq =%6.2f, Tag =%3d",
    Qstr[plttype-1], source,selfreq,tag);
 strcpy(in_file,file_name);
 strcat(in_file,rq);
 if ((infile = efopen (in_file, "r")) == NULL)
  return (0);

 /* dump first 6 lines */
 fgets(line, MAXLINE, infile);
 fgets(line, MAXLINE, infile);
 fgets(line, MAXLINE, infile);
 fgets(line, MAXLINE, infile);
```

```
  fgets(line, MAXLINE, infile);
  fgets(line, MAXLINE, infile);

  /* now read in data */
  i = 1;
  while ((fscanf(infile,"%5d%c %d %f %f %f %f %f %d",
        &ia,&ch,&ib,&a,&b,&c,&d,&e,&ic)) != EOF) {
   if ((source == ic)&&(selfreq == e)&&((tag == ib)||(tag == 0))) {
    if (ch == 'E') i=i-1;
    switch (plttype) {
     case 1: plotData->yarray[i] = d;
          plotData->xarray[i] = a;
          break;
     case 2: plotData->yarray[i] = d;
          plotData->xarray[i] = b;
          break;
     case 3: plotData->yarray[i] = d;
          plotData->xarray[i] = c;
          break;
     case 4: plotData->yarray[i] = d;
          plotData->xarray[i] = (float) ia;
          break;
    }
    if (i == 1) {
     plotData->xlunit = plotData->xarray[1];
     plotData->xrunit = plotData->xarray[1];
     plotData->ytunit = plotData->yarray[1];
     plotData->ybunit = plotData->yarray[1];
    } else {
     if (plotData->xarray[i] < plotData->xlunit)
      plotData->xlunit = plotData->xarray[i];
     if (plotData->xarray[i] > plotData->xrunit)
      plotData->xrunit = plotData->xarray[i];
     if (plotData->yarray[i] < plotData->ybunit)
      plotData->ybunit = plotData->yarray[i];
     if (plotData->yarray[i] > plotData->ytunit)
      plotData->ytunit = plotData->yarray[i];
    }
    i++;
   }
  }
  fclose(infile);
  plotData->numxpts = i-1;
  if (i-1 == 0) {
   createMessageDialog (topLevel, "Charges",
     "Graph contains no data points.", XmDIALOG_MESSAGE);
   return (-1);
  } else
   return (0);
}

static int getCP (drawing_a)
 Widget      drawing_a;
{
 static char CPstr[] = "Isolation vs Frequency";
 int i;
 float a,b;
 PlotData *plotData;
 extern FILE *efopen ();

 /* Get data structure from widget for storing plot data */
 XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);

 sprintf(plotData->titlestr,"%s",CPstr);
 strcpy(in_file,file_name);
 strcat(in_file,rcp);
 if ((infile = efopen (in_file, "r")) == NULL)
  return (0);

 /* dump first 4 lines */
 fgets(line, MAXLINE, infile);
 fgets(line, MAXLINE, infile);
 fgets(line, MAXLINE, infile);
 fgets(line, MAXLINE, infile);

 /* now read in data */
 i = 1;
 while ((fscanf(infile,"%f %f",&a,&b)) != EOF) {
  plotData->yarray[i] = b;
  plotData->xarray[i] = a;
  if (i == 1) {
   plotData->xlunit = plotData->xarray[1];
   plotData->xrunit = plotData->xarray[1];
   plotData->ytunit = plotData->yarray[1];
   plotData->ybunit = plotData->yarray[1];
  } else {
   if (plotData->xarray[i] < plotData->xlunit)
    plotData->xlunit = plotData->xarray[i];
   if (plotData->xarray[i] > plotData->xrunit)
    plotData->xrunit = plotData->xarray[i];
   if (plotData->yarray[i] < plotData->ybunit)
    plotData->ybunit = plotData->yarray[i];
   if (plotData->yarray[i] > plotData->ytunit)
```

```
        plotData->ytunit = plotData->yarray[i];
      }
      i++;
    }
    fclose(infile);
    plotData->numxpts = i-1;
    if (i-1 == 0) {
      createMessageDialog (topLevel, "Coupling",
        "Graph contains no data points.", XmDIALOG_MESSAGE);
      return (-1);
    } else
      return (0);
}

static int getPT (drawing_a)
    Widget      drawing_a;
{
    static char *PTstr[4] = {
      "Vertical vs Theta",
      "Horizontal vs Theta",
      "Vertical vs Phi",
      "Horizontal vs Phi"};
    char lookfor1[25];
    char lookfor2[15];
    int i;
    float a,b,c,d,e,f,g,h;
    PlotData *plotData;
    extern FILE *efopen ();

    /* Get data structure from widget for storing plot data */
    XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);

    if ((plttype == 1)||(plttype == 2))
      sprintf(plotData->titlestr,"%s, Source =%5d, Freq =%6.2f, Phi =%6.1f",
          PTstr[plttype-1],source,selfreq,angle);
    else
      sprintf(plotData->titlestr,"%s, Source =%5d, Freq =%6.2f, Theta =%6.1f",
          PTstr[plttype-1],source,selfreq,angle);
    strcpy(in_file,file_name);
    strcat(in_file,rpt);
    if ((infile = efopen (in_file, "r")) == NULL)
      return (-1);

    /* first look for frequency */
    sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
    fgets(line, MAXLINE, infile);
    *(line + 22) = '\0';
    while(strcmp(lookfor1,line) != 0) {
      if (fgets(line, MAXLINE, infile) == NULL) {
        createMessageDialog (topLevel, "Radiation Patterns",
          "Could not plot.  Frequency not found.", XmDIALOG_MESSAGE);
        return (-1);
      }
      *(line + 22) = '\0';
    }
    /* now look for source */
    sprintf(lookfor2," SOURCE :%5d",source);
    fgets(line, MAXLINE, infile);
    *(line + 14) = '\0';
    while(strcmp(lookfor2,line) != 0) {
      if (fgets(line, MAXLINE, infile) == NULL) {
        createMessageDialog (topLevel, "Radiation Patterns",
          "Could not plot.  Source not found.", XmDIALOG_MESSAGE);
        return (-1);
      }
      *(line + 14) = '\0';
    }

    /* now read in data */
    fgets(line, MAXLINE, infile);
    fgets(line, MAXLINE, infile);
    i = 1;
    while ((fscanf(infile,"%f %f %f %f %f %f %f %f",
            &a,&b,&c,&d,&e,&f,&g,&h)) == 8) {
      if ((((plttype == 1)||(plttype == 2))&&(angle == b))||
          (((plttype == 3)||(plttype == 4))&&(angle == a))) {
        switch (plttype) {
          case 1:  plotData->yarray[i] = c;
                   plotData->xarray[i] = a;
                   break;
          case 2:  plotData->yarray[i] = d;
                   plotData->xarray[i] = a;
                   break;
          case 3:  plotData->yarray[i] = c;
                   plotData->xarray[i] = b;
                   break;
          case 4:  plotData->yarray[i] = d;
                   plotData->xarray[i] = b;
                   break;
        }
        if (i == 1) {
          plotData->xlunit = plotData->xarray[1];
          plotData->xrunit = plotData->xarray[1];
```

```
        plotData->ytunit = plotData->yarray[1];
        plotData->ybunit = plotData->yarray[1];
      } else {
        if (plotData->xarray[i] < plotData->xlunit)
          plotData->xlunit = plotData->xarray[i];
        if (plotData->xarray[i] > plotData->xrunit)
          plotData->xrunit = plotData->xarray[i];
        if (plotData->yarray[i] < plotData->ybunit)
          plotData->ybunit = plotData->yarray[i];
        if (plotData->yarray[i] > plotData->ytunit)
          plotData->ytunit = plotData->yarray[i];
      }
      i++;
    }
  }
  fclose(infile);
  plotData->numxpts = i-1;
  if (i-1 == 0) {
    createMessageDialog (topLevel, "Radiation Patterns",
      "Graph contains no data points.", XmDIALOG_MESSAGE);
    return (-1);
  } else
    return (0);
}

static int getNE (drawing_a)
  Widget      drawing_a;
{
  static char *NEstr[8] = {
    "Ez_nor vs X",
    "Ez_nor vs Y",
    "Ez_nor vs Z",
    "Ez_nor vs Frequency",
    "E_nor vs X",
    "E_nor vs Y",
    "E_nor vs Z",
    "E_nor vs Frequency"};
  char lookfor1[22];
  char lookfor2[14];
  char fline[30];
  int i;
  float curfreq;
  float a,b,c,d,e,o,p;
  PlotData *plotData;

  /* Get data structure from widget for storing plot data */
  XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);

  if ((plttype == 1)||(plttype == 5))
    sprintf(plotData->titlestr,"%s, Source =%5d, Y =%6.2f, Z =%6.2f, Freq =%6.2f",
        NEstr[plttype-1],source,sely,selz,selfreq);
  else if ((plttype == 2)||(plttype == 6))
    sprintf(plotData->titlestr,"%s, Source =%5d, X =%6.2f, Z =%6.2f, Freq =%6.2f",
        NEstr[plttype-1],source,selx,selz,selfreq);
  else if ((plttype == 3)||(plttype == 7))
    sprintf(plotData->titlestr,"%s, Source =%5d, X =%6.2f, Y =%6.2f, Freq =%6.2f",
        NEstr[plttype-1],source,selx,sely,selfreq);
  else if ((plttype == 4)||(plttype == 8))
    sprintf(plotData->titlestr,"%s, Source =%5d, X =%6.2f, Y =%6.2f, Z =%6.2f",
        NEstr[plttype-1],source,selx,sely,selz);
  i = 1;
  strcpy(in_file,file_name);
  strcat(in_file,me);
  if ((infile = fopen(in_file,"r")) == (FILE*)0) {
    printf("\n\t\t File %s could not be opened. \n", in_file);
    exit(1);
  }
  /* handle plot types 4 and 8 differently from the rest */
  if ((plttype == 4)||(plttype == 8)) {
  /* look for a frequency line */
    sprintf(lookfor1," FREQUENCY =");
    while (fgets(line, MAXLINE, infile) != NULL) {
      *(line + 12) = '\0';
      if (strcmp(lookfor1, line) == 0) {
        strcpy(fline,line+13);
        printf("3:fline is:%s...",fline);
        *(fline + 9) = '\0';
        curfreq = atof(fline);

        /* now look for currect source */
        sprintf(lookfor2," SOURCE :%5d",source);
        fgets(line, MAXLINE, infile);
        *(line + 14) = '\0';
        while(strcmp(lookfor2,line) != 0)
        {
          if (fgets(line, MAXLINE, infile) == NULL) {
            createMessageDialog (topLevel, "Near Electric Fields",
              "Could not plot.  Source not found.", XmDIALOG_MESSAGE);
            return (-1);
          }
          *(line + 14) = '\0';
        }
```

```
    /* now discard next two lines */
    fgets(line, MAXLINE, infile);
    fgets(line, MAXLINE, infile);

    /* now scan through data looking for match to x,y,and z */
    while ((fscanf(infile,"%f %f %f %f %f %f %f",
              &a,&b,&c,&d,&e,&o,&p)) == 7) {
     if ((a==selx)&&(b==sely)&&(c==selz)) {
      plotData->xarray[i] = curfreq;
      if (plttype == 4) plotData->yarray[i] = o;
      if (plttype == 8) plotData->yarray[i] = p;
      if (i == 1) {
       plotData->xlunit = plotData->xarray[1];
       plotData->xrunit = plotData->xarray[1];
       plotData->ytunit = plotData->yarray[1];
       plotData->ybunit = plotData->yarray[1];
      } else {
       if (plotData->xarray[i] < plotData->xlunit)
        plotData->xlunit = plotData->xarray[i];
       if (plotData->xarray[i] > plotData->xrunit)
        plotData->xrunit = plotData->xarray[i];
       if (plotData->yarray[i] < plotData->ybunit)
        plotData->ybunit = plotData->yarray[i];
       if (plotData->yarray[i] > plotData->ytunit)
        plotData->ytunit = plotData->yarray[i];
      }
      i++;
     }
    }
   }
  }
 }
 else {
 /* first look for frequency */
 sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
 fgets(line, MAXLINE, infile);
 *(line + 22) = '\0';
 while(strcmp(lookfor1,line) != 0) {
  if (fgets(line, MAXLINE, infile) == NULL) {
   createMessageDialog (topLevel, "Near Electric Fields",
     "Could not plot.  Frequency not found.", XmDIALOG_MESSAGE);
   return (-1);
  }
  *(line + 22) = '\0';
 }
 /* now look for source */
 sprintf(lookfor2," SOURCE :%5d",source);
 fgets(line, MAXLINE, infile);
 *(line + 14) = '\0';
 while(strcmp(lookfor2,line) != 0) {
  if (fgets(line, MAXLINE, infile) == NULL) {
   createMessageDialog (topLevel, "Near Electric Fields",
     "Could not plot.  Source not found.", XmDIALOG_MESSAGE);
   return (-1);
  }
  *(line + 14) = '\0';
 }

 /* now read in data */
 fgets(line, MAXLINE, infile);
 fgets(line, MAXLINE, infile);
 i = 1;
 while ((fscanf(infile,"%f %f %f %f %f %f %f",
           &a, &b, &c, &d, &e, &o, &p)) == 7) {
  if ((((plttype == 1)||(plttype == 5))&&(b == sely)&&(c == selz))||
     (((plttype == 2)||(plttype == 6))&&(a == selx)&&(c == selz))||
     (((plttype == 3)||(plttype == 7))&&(a == selx)&&(b == sely))) {
   switch (plttype) {
    case 1:  plotData->yarray[i] = o;
          plotData->xarray[i] = a;
          break;
    case 2:  plotData->yarray[i] = o;
          plotData->xarray[i] = b;
          break;
    case 3:  plotData->yarray[i] = o;
          plotData->xarray[i] = c;
          break;
    case 5:  plotData->yarray[i] = p;
          plotData->xarray[i] = a;
          break;
    case 6:  plotData->yarray[i] = p;
          plotData->xarray[i] = b;
          break;
    case 7:  plotData->yarray[i] = p;
          plotData->xarray[i] = c;
          break;
   }
   if (i == 1) {
    plotData->xlunit = plotData->xarray[1];
    plotData->xrunit = plotData->xarray[1];
    plotData->ytunit = plotData->yarray[1];
    plotData->ybunit = plotData->yarray[1];
   } else {
```

```
        if (plotData->xarray[i] < plotData->xlunit)
          plotData->xlunit = plotData->xarray[i];
        if (plotData->xarray[i] > plotData->xrunit)
          plotData->xrunit = plotData->xarray[i];
        if (plotData->yarray[i] < plotData->ybunit)
          plotData->ybunit = plotData->yarray[i];
        if (plotData->yarray[i] > plotData->ytunit)
          plotData->ytunit = plotData->yarray[i];
      }
      i++;
    }
  }
  }
  fclose(infile);
  plotData->numxpts = i-1;
  if (i-1 == 0) {
    createMessageDialog (topLevel, "Near Electric Fields",
      "Graph contains no data points.", XmDIALOG_MESSAGE);
    return (-1);
  } else
    return (0);
}

static int getNH (drawing_a)
  Widget      drawing_a;
{
  static char *NHstr[8] = {
    "Hx_nor vs X",
    "Hx_nor vs Y",
    "Hx_nor vs Z",
    "Hx_nor vs Frequency",
    "Hy_nor vs X",
    "Hy_nor vs Y",
    "Hy_nor vs Z",
    "Hy_nor vs Frequency"};
  char lookfor1[22];
  char lookfor2[14];
  char fline[30];
  int i;
  float curfreq;
  float a,b,c,d,e,o,p;
  PlotData *plotData;

  /* Get data structure from widget for storing plot data */
  XtVaGetValues (drawing_a, XmNuserData, &plotData, NULL);

  if ((plttype == 1)||(plttype == 5))
    sprintf(plotData->ttlestr,"%s, Source =%5d, Y =%6.2f, Z =%6.2f, Freq =%6.2f",
        NHstr[plttype-1],source,sely,selz,selfreq);
  else if ((plttype == 2)||(plttype == 6))
    sprintf(plotData->ttlestr,"%s, Source =%5d, X =%6.2f, Z =%6.2f, Freq =%6.2f",
        NHstr[plttype-1],source,selx,selz,selfreq);
  else if ((plttype == 3)||(plttype == 7))
    sprintf(plotData->ttlestr,"%s, Source =%5d, X =%6.2f, Y =%6.2f, Freq =%6.2f",
        NHstr[plttype-1],source,selx,sely,selfreq);
  else if ((plttype == 4)||(plttype == 8))
    sprintf(plotData->ttlestr,"%s, Source =%5d, X =%6.2f, Y =%6.2f, Z =%6.2f",
        NHstr[plttype-1],source,selx,sely,selz);
  i = 1;
  strcpy(in_file,file_name);
  strcat(in_file,mm);
  if ((infile = fopen(in_file,"r")) == (FILE*)0) {
    printf("\n\t\t File %s could not be opened. \n", in_file);
    exit(1);
  }
  /* handle plot types 4 and 8 differently from the rest */
  if ((plttype == 4)||(plttype == 8)) {
  /* look for a frequency line */
    sprintf(lookfor1," FREQUENCY =");
    while (fgets(line, MAXLINE, infile) != NULL) {
      *(line + 12) = '\0';
      if (strcmp(lookfor1, line) == 0) {
        strcpy(fline,line+13);
        printf("3:fline is:%s...",fline);
        *(fline + 9) = '\0';
        curfreq = atof(fline);

        /* now look for currect source */
        sprintf(lookfor2," SOURCE :%5d",source);
        fgets(line, MAXLINE, infile);
        *(line + 14) = '\0';
        while(strcmp(lookfor2,line) != 0)
        {
          if (fgets(line, MAXLINE, infile) == NULL) {
            createMessageDialog (topLevel, "Near Magnetic Fields",
              "Could not plot.  Source not found.", XmDIALOG_MESSAGE);
            return (-1);
          }
          *(line + 14) = '\0';
        }

        /* now discard next two lines */
        fgets(line, MAXLINE, infile);
```

```
    fgets(line, MAXLINE, infile);

    /* now scan through data looking for match to x,y,and z */
    while ((fscanf(infile,"%f %f %f %f %f %f",
              &a,&b,&c,&d,&e,&o,&p)) == 7) {
      if ((a==sebx)&&(b==sely)&&(c==selz)) {
        plotData->xarray[i] = curfreq;
        if (plttype == 4) plotData->yarray[i] = d;
        if (plttype == 8) plotData->yarray[i] = e;
        if (i == 1) {
          plotData->xlunit = plotData->xarray[1];
          plotData->xrunit = plotData->xarray[1];
          plotData->ytunit = plotData->yarray[1];
          plotData->ybunit = plotData->yarray[1];
        } else {
          if (plotData->xarray[i] < plotData->xlunit)
            plotData->xlunit = plotData->xarray[i];
          if (plotData->xarray[i] > plotData->xrunit)
            plotData->xrunit = plotData->xarray[i];
          if (plotData->yarray[i] < plotData->ybunit)
            plotData->ybunit = plotData->yarray[i];
          if (plotData->yarray[i] > plotData->ytunit)
            plotData->ytunit = plotData->yarray[i];
        }
        i++;
      }
    }
  }
 }
}
else {
/* first look for frequency */
sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
fgets(line, MAXLINE, infile);
*(line + 22) = '\0';
while(strcmp(lookfor1,line) != 0) {
 if (fgets(line, MAXLINE, infile) == NULL) {
  createMessageDialog (topLevel, "Near Magnetic Fields",
    "Could not plot.  Frequency not found.", XmDIALOG_MESSAGE);
  return (-1);
 }
 *(line + 22) = '\0';
}
/* now look for source */
sprintf(lookfor2," SOURCE :%5d",source);
fgets(line, MAXLINE, infile);
*(line + 14) = '\0';
while(strcmp(lookfor2,line) != 0) {
 if (fgets(line, MAXLINE, infile) == NULL) {
  createMessageDialog (topLevel, "Near Magnetic Fields",
    "Could not plot.  Source not found.", XmDIALOG_MESSAGE);
  return (-1);
 }
 *(line + 14) = '\0';
}

/* now read in data */
fgets(line, MAXLINE, infile);
fgets(line, MAXLINE, infile);
i = 1;
while ((fscanf(infile,"%f %f %f %f %f %f",
          &a, &b, &c, &d, &e, &o, &p)) == 7) {
 if ((((plttype == 1)||(plttype == 5))&&(b == sely)&&(c == selz))||
    (((plttype == 2)||(plttype == 6))&&(a == selx)&&(c == selz))||
    (((plttype == 3)||(plttype == 7))&&(a == sebx)&&(b == sely))) {
  switch (plttype) {
    case 1:  plotData->yarray[i] = d;
          plotData->xarray[i] = a;
          break;
    case 2:  plotData->yarray[i] = d;
          plotData->xarray[i] = b;
          break;
    case 3:  plotData->yarray[i] = d;
          plotData->xarray[i] = c;
          break;
    case 5:  plotData->yarray[i] = e;
          plotData->xarray[i] = a;
          break;
    case 6:  plotData->yarray[i] = e;
          plotData->xarray[i] = b;
          break;
    case 7:  plotData->yarray[i] = e;
          plotData->xarray[i] = c;
          break;
  }
  if (i == 1) {
    plotData->xlunit = plotData->xarray[1];
    plotData->xrunit = plotData->xarray[1];
    plotData->ytunit = plotData->yarray[1];
    plotData->ybunit = plotData->yarray[1];
  } else {
    if (plotData->xarray[i] < plotData->xlunit)
      plotData->xlunit = plotData->xarray[i];
```

```
      if (plotData->xarray[i] > plotData->xrunit)
        plotData->xrunit = plotData->xarray[i];
      if (plotData->yarray[i] < plotData->ybunit)
        plotData->ybunit = plotData->yarray[i];
      if (plotData->yarray[i] > plotData->ytunit)
        plotData->ytunit = plotData->yarray[i];
    }
    i++;
  }
 }
 }
 fclose(infile);
 plotData->numxpts = i-1;
 if (i-1 == 0) {
  createMessageDialog (topLevel, "Near Magnetic Fields",
    "Graph contains no data points.", XmDIALOG_MESSAGE);
  return (-1);
 } else
  return (0);
}

static void drawing_linear(widget)
   Widget   widget;
{
 int          i, x, y, xwidth, ywidth, ixs, iys;
 int  ixpos, iypos, iy0, iy1, ix0, ix1, deltaxpix, deltaypix;
 float xs, ys, xrange, yrange;
 float xdiv, ydiv, xinc, yinc, xpos, ypos;
 float xunit, yunit, deltax, deltay;
 char  labelstr[11];
 PlotData *plotData;

 XtVaGetValues(widget, XmNuserData, &plotData, NULL);

 SRGP_setColor (2);
 x=100;
 y=100;
 xwidth=500;
 ywidth=500;
 SRGP_rectangleCoord (x, y, x+xwidth, y+ywidth);

 xdiv=10;
 ydiv=5;
 xinc=(600-100)/xdiv;
 xpos=x;
 iy0=100;
 iy1=600;
 for (i=1; i<=xdiv-1; i++) {
  xpos=xpos+xinc;
  ixpos=xpos;
  SRGP_lineCoord (ixpos, 700-iy0, ixpos, 700-iy1);
 }

 yinc=(600-100)/ydiv;
 ypos=y;
 ix0=100;
 ix1=600;
 for (i=1; i<=ydiv-1; i++) {
  ypos=ypos+yinc;
  iypos=ypos;
  SRGP_lineCoord (ix0, 700-iypos, ix1, 700-iypos);
 }

 /*  x unit label  */
 iy1 = 620;
 deltax = (plotData->xrunit - plotData->xlunit) / xdiv;
 deltaxpix = 50;
 xpos = 100 - deltaxpix - 15;
 xunit = plotData->xlunit;
 for (i=1; i<= xdiv+1; i++) {
  xpos=xpos+deltaxpix;
  ixpos=xpos;
  sprintf(labelstr, "%5f", xunit);
  labelstr[5] = '\0';
  SRGP_text (SRGP_defPoint(ixpos, 700-iy1), labelstr);
  xunit=plotData->xlunit+(i*deltax);
 }

 /*  y unit label  */
 ix1=60;
 deltay=(plotData->ytunit-plotData->ybunit)/ydiv;
 deltaypix=100;
 ypos=100-deltaypix+5;
 yunit=plotData->ytunit;
 for (i=1; i<= ydiv+1; i++) {
  ypos=ypos+deltaypix;
  iypos=ypos;
  sprintf(labelstr, "%5f", yunit);
  labelstr[5] = '\0';
  SRGP_text (SRGP_defPoint(ix1, 700-iypos), labelstr);
  yunit=plotData->ytunit-(i*deltay);
 }
```

```
 /* draw linear title */
 x=350-strlen(plotData->titlestr)*4;
 y=50;
 SRGP_text(SRGP_defPoint(x, 700-y), plotData->titlestr);


 /* draw linear curve */
 SRGP_setColor (4);
 SRGP_setLineWidth (3);
 xrange=plotData->xrunit-plotData->xlunit;
 yrange=plotData->ytunit-plotData->ybunit;
 for (i=1; i<=plotData->numxpts; i++) {
  xs = ((plotData->xarray[i]-plotData->xlunit)/xrange)*(600-100)+100;
  ys = -1*(((plotData->yarray[i]-plotData->ytunit)/yrange)*(600-100)-100);
  ixs=xs;
  iys=ys;
  if (i==1) { x=xs; y=ys; }
  SRGP_lineCoord(x, 700-y, ixs, 700-iys);
  x=xs;
  y=ys;
 }
}


static void drawing_smith(widget)
   Widget    widget;
{
 int           i, x, y, x2, y2, ixs, iys;
 float vswrc_radius, vswrc;
 float vyt = 100, vyb = 600, vxl = 100, vxr = 600, wyt = 1.0, wyb = -1.0,
    wxl = -1.0, wxr = 1.0;
 float rr,aa,bb,cc,dd,xw,yw,xs,ys;
 char freqstr[10];
 PlotData *plotData;

 XtVaGetValues(widget, XmNuserData, &plotData, NULL);

 SRGP_setColor (2);
 x=350;
 y=100;
 x2=350;
 y2=600;
 SRGP_lineCoord(x, 700-y, x2, 700-y2);

 for (i=100; i<350; i+=62.5) {
  x=i;
  if (i==100) y=i; else y=y+2*62.5;
  x2=700-x*2;
  y2=x2;
  SRGP_ellipse (SRGP_defRectangle(x, 700-(y+y2), x+x2, 700-y));
 }
 x = -150;
 y=350;
 SRGP_ellipseArc (SRGP_defRectangle(x, 700-(y+500), x+500, 700-y), 0, 90);
 x=350;
 y=350;
 SRGP_ellipseArc (SRGP_defRectangle(x, 700-(y+500), x+500, 700-y), 90, 180);
 vswrc=3;
 vswrc_radius=500.0*(((2*vswrc)/(vswrc+1))-1.0);
 x=vswrc_radius;
 y=vswrc_radius;
 SRGP_ellipse (SRGP_defRectangle(350-x/2, 700-(350+y/2), 350+x/2, 700-(350-y/2)));
 x=0;
 y=50;
 SRGP_text(SRGP_defPoint(x, 700-y), plotData->titlestr);

 /* draw 0, 90, 180, 270 degree marks */
 x=620;
 y=355;
 SRGP_text(SRGP_defPoint(x, 700-y), "+1");
 x=348;
 y=90;
 SRGP_text(SRGP_defPoint(x, 700-y), "0");
 x=70;
 y=355;
 SRGP_text(SRGP_defPoint(x, 700-y), "-1");
 x=345;
 y=620;
 SRGP_text(SRGP_defPoint(x, 700-y), "00");

 /* change color for impedance curve */
 SRGP_setColor (4);
 SRGP_setLineWidth (3);

 for (i=1; i<=plotData->numxpts; i++) {
  /* smith chart to pixel screen transformation */
  sprintf(freqstr,"%6.2f",plotData->freq[i]);
  rr=plotData->xarray[i];
  aa=rr-1;
  bb=plotData->yarray[i];
  cc=rr+1;
  dd=bb;
  yw = -(aa*cc+bb*dd)/(cc*cc+dd*dd);
```

```
    xw=(bb*cc-aa*dd)/(cc*cc+dd*dd);
    xs=((vxr-vxl)/(wxr-wxl))*(xw-wxl)+vxl;
    ys=((vyt-vyb)/(wyt-wyb))*(yw-wyb)+vyb;
    ixs=xs;
    iys=ys;
    if (i==1)
      { x=xs; y=ys; }
    SRGP_lineCoord(x, 700-y, ixs, 700-iys);
    SRGP_text(SRGP_defPoint(ixs, 700-iys), freqstr);
    x=xs;
    y=ys;
  }
}

static void drawing_polar(widget)
 Widget      widget;
{
 int            i, x, y, x2, y2, lx, ly, ixs, iys, idbmin;
 float xs, ys, a, r, absdb;
 float dbouter, dbinc, dbmin;
 char dbstr[10];
 PlotData *plotData;

 XtVaGetValues(widget, XmNuserData, &plotData, NULL);

 SRGP_setColor (2);
 x=350;
 y=100;
 x2=350;
 y2=600;
 SRGP_lineCoord(x, 700-y, x2, 700-y2);
 x=100;
 y=350;
 x2=600;
 y2=350;
 SRGP_lineCoord(x, 700-y, x2, 700-y2);

 /* draw center db label */
 lx=350;
 ly=365;
 dbouter=20;
 dbinc=10;
 dbmin=dbouter - 5 * dbinc;
 idbmin=dbmin;
 sprintf(dbstr,"%3d",idbmin);
 SRGP_text(SRGP_defPoint(lx, 700-ly), dbstr);

 for (i=100; i<350; i+=50) {
  x=i;
  y=i;
  x2=700-x*2;
  y2=x2;
  SRGP_ellipse (SRGP_defRectangle(x, 700-(y+y2), x+x2, 700-y));
  idbmin=idbmin + dbinc;
  sprintf(dbstr,"%3d",idbmin);
  lx=50+x2/2;
  SRGP_text(SRGP_defPoint(lx, 700-ly), dbstr);
 }
 dbmin=dbouter-5*dbinc;
 x=350-strlen(plotData->titlestr)*4;
 y=50;
 SRGP_text(SRGP_defPoint(x, 700-y), plotData->titlestr);

 /* draw 0, 90, 180, 270 degree marks */
 x=620;
 y=355;
 SRGP_text(SRGP_defPoint(x, 700-y), "0");
 x=345;
 y=90;
 SRGP_text(SRGP_defPoint(x, 700-y), "90");
 x=70;
 y=355;
 SRGP_text(SRGP_defPoint(x, 700-y), "180");
 x=340;
 y=620;
 SRGP_text(SRGP_defPoint(x, 700-y), "270");

 /* Draw polar plot */
 SRGP_setColor (4);
 SRGP_setLineWidth (3);
 for (i=1; i<=plotData->numxpts; i++) {
  a=360-plotData->xarray[i];
  a=a*(3.14159/180);
  absdb=dbmin-dbouter;
  if (absdb < 0) absdb*=-1;
  if (plotData->yarray[i] < dbmin)
    r=0;
  else
    r=(plotData->yarray[i]-dbmin) / absdb;
  r=r*((600-100)/2);
  xs=r*cos(a)+350;
  ys=r*sin(a)+350;
  ixs=xs;
```

```
    iys=ys;
    if (i==1) {x=xs; y=ys;}
    SRGP_lineCoord(x, 700-y, ixs, 700-iys);
    x=xs;
    y=ys;
   }
  }

  /* Callback to deallocate PlotData structure upon destruction of window.
   */
  static void destroyCB (widget)
   Widget   widget;
  {
   PlotData *plotData;

   XtVaGetValues (widget, XmNuserData, &plotData, NULL);
   XtFree ((char *) plotData);

  } /* end destroyCB */
```

A.12 cOutputMenu.c:

cOutputMenu.c:

```
/*
 * Filename : cOutputMenu.c
 *
 * Callbacks for the Output menu items.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>        /* For using stat() */
#include <sys/stat.h>
#include <X11/IntrinsicP.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/PanedW.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include "actionArea.h"
#include "filter.h"

int sourceData[100];
float freqData[100];
int numFreqData;
float phiData[100], thetaData[100];
int numPhiData, numThetaData;
float xData[100], yData[100], zData[100];
int numXData, numYData, numZData;

extern char    *necOutputFilename, *necInputFilename;

extern void    createMessageDialog ();
extern Boolean          isPoppedUp ();
extern void    newEscapeAction();
extern Widget  createScrolledText();
extern void    editScrolledText ();
extern void    extract ();
extern int     necRunStatus ();
extern void    needsfit ();

static Widget  plotDialog = NULL, powerText, sourceText, freqText,
           tagText, angleText, sel1Text, sel2Text,
           sel3Text, angleLabel, sel1Label, sel2Label, sel3Label;
static int     plotType;

Boolean        alreadyFiltered ();
static void    cancelButtonCB ();
static void    createOptionMenu ();
static void    destroyDialogCB ();
static void    plotButtonCB ();
static void    plotTypeCB ();

static enum Options optionType;    /* Impedance, Admittance, Currents, ... */

typedef struct _menu_item {
  char        *name;
  void        (*callback) ();
} MenuItem;

static MenuItem impedancePlotTypes [] = {
  {"Resistance vs. Frequency"},
  {"Reactance vs. Frequency"},
  {"Smith Chart"},
  NULL
};

static MenuItem admittancePlotTypes [] = {
  {"Conductance vs. Wavelength"},
  {"Susceptance vs. Wavelength"},
  NULL
};

static MenuItem currentsPlotTypes [] = {
  {"Magnitude vs. X"},
  {"Magnitude vs. Y"},
  {"Magnitude vs. Z"},
  {"Magnitude vs. Segment"},
  {"Phase vs. X"},
  {"Phase vs. Y"},
  {"Phase vs. Z"},
  {"Phase vs. Segment"},
  NULL
};

static MenuItem chargePlotTypes [] = {
  {"Charge vs. X"},
  {"Charge vs. Y"},
```

```
  {"Charge vs. Z"},
  {"Charge vs. Segment"},
  NULL
};

static MenuItem couplingPlotTypes [] = {
  {"Isolation vs. Frequency"},
  NULL
};

static MenuItem nearElectricPlotTypes [] = {
  {"Ez Normalized vs. X"},
  {"Ez Normalized vs. Y"},
  {"Ez Normalized vs. Z"},
  {"Ez Normalized vs. Frequency"},
  {"E Normalized vs. X"},
  {"E Normalized vs. Y"},
  {"E Normalized vs. Z"},
  {"E Normalized vs. Frequency"},
  NULL
};

static MenuItem nearMagneticPlotTypes [] = {
  {"Hx Normalized vs. X"},
  {"Hx Normalized vs. Y"},
  {"Hx Normalized vs. Z"},
  {"Hx Normalized vs. Frequency"},
  {"Hy Normalized vs. X"},
  {"Hy Normalized vs. Y"},
  {"Hy Normalized vs. Z"},
  {"Hy Normalized vs. Frequency"},
  NULL
};

static MenuItem radiationPlotTypes [] = {
  {"Vertical vs. Theta"},
  {"Horizontal vs. Theta"},
  {"Vertical vs. Phi"},
  {"Horizontal vs. Phi"},
  NULL
};

static MenuItem *optionTypes [] = {
  impedancePlotTypes,
  admittancePlotTypes,
  currentsPlotTypes,
  chargePlotTypes,
  couplingPlotTypes,
  nearElectricPlotTypes,
  nearMagneticPlotTypes,
  radiationPlotTypes
};

/*****************************************************************
 * Opens a dialog box to query user for plotting data - plot type,
 * source, frequency, tag & theta.
 */
void openPlotDialog (w, cType)
  Widget      w;
  char        *cType;
{
  Widget      form, pane, rowColumn, label, actionA;
  int         n;
  char        title [50];
  Arg         args[13];
  XmString    string;
  Boolean     bPower  = False,
              bSource = False,
              bFreq   = False,
              bTag    = False,
              bAngle  = False,
              bXYZfreq= False,
                                                    bXYZfreq2= False,
              tryFilter = True;
  extern Widget topLevel;
  static ActionAreaItem actionItems[] = {
    {"Plot",   plotButtonCB,    NULL},
    {"Cancel", cancelButtonCB,  NULL},
  };
  static char  *titleTypes [] = {
    "Impedance", "Admittance", "Currents", "Charge", "Coupling",
    "Near Electric Fields", "Near Magnetic Fields", "Radiation Patterns"
    };
  extern int *EX_Wire;

  /* Check NEC run status before continuing */
  if (!necRunStatus()) return;

  /* Make sure that the user has entered a NEC Output filename */
  if (necOutputFilename == NULL) {
    createMessageDialog (topLevel, "Plot Error",
      "NEC Output file must be specified", XmDIALOG_ERROR);
    return;
```

```
}

if (plotDialog != NULL && !isPoppedUp(plotDialog)) {
  XtDestroyWidget(plotDialog);
  plotDialog = NULL;
}

/* Allow only one plot dialog to be opened at a time */
if (plotDialog != NULL) return;

optionType = (enum Options) atoi (cType);
plotType = 1;
switch (optionType) {
  case CURRENTS:
    tryFilter = False;
    bPower = True;
    bSource = True;
    bFreq = True;
    bTag = True;
    break;
  case CHARGE:
    tryFilter = False;
    bPower = True;
    bSource = True;
    bFreq = True;
    bTag = True;
    break;
  case RADIATION:
    bSource = True;
    bFreq = True;
    bAngle = True;
    break;
  case IMPEDANCE:
    bSource = True;
    break;
  case ADMITTANCE:
    bSource = True;
    break;
  case NEAR_ELECTRIC:
    tryFilter = False;
    bPower = True;
    bSource = True;
    bXYZfreq = True;
    break;
  case NEAR_MAGNETIC:
    tryFilter = False;
    bPower = True;
    bSource = True;
    bXYZfreq2 = True;
  default:
    break;
}

if (tryFilter) {
  if (!alreadyFiltered (fileExts[optionType])) {
    needsflt (optionType, 0);
  }
}

sprintf (title, "Plotting Options for %s", titleTypes[optionType]);
plotDialog = XtVaCreatePopupShell (NULL,
    topLevelShellWidgetClass, topLevel,
    XmNtitle,          title,
    XmNallowShellResize,  True,
    XmNdeleteResponse,    XmDESTROY,
    NULL);

newEscapeAction(plotDialog);

XtAddCallback (plotDialog, XmNdestroyCallback, destroyDialogCB, NULL);

n = 0;
XtSetArg (args[n], XmNsashWidth, 1); n++;
XtSetArg (args[n], XmNsashHeight, 1); n++;
pane = XmCreatePanedWindow (plotDialog, "pane", args, n);
XtManageChild (pane);

form = XmCreateForm (pane, "form", NULL, 0);

n = 0;
XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNrightOffset, 10); n++;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNtopOffset, 10); n++;
XtSetArg (args [n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNbottomOffset, 10); n++;
rowColumn = XmCreateRowColumn (form, "rowColumn", args, n);
XtManageChild (rowColumn);

createOptionMenu (rowColumn, optionTypes [optionType]);

XtSetArg (args [0], XmNeditMode, XmSINGLE_LINE_EDIT);
XtSetArg (args [1], XmNcolumns, 11);
```

```
if (bPower) {
 powerText = XmCreateText (rowColumn, "text", args, 2);
 XtManageChild (powerText);

 XmTextSetString(powerText, "1000");
}
if (bSource) {
 extern int VoltageSourcesCount;
 int index;

 for (index = 0; index < VoltageSourcesCount; index++)
  sourceData[index] = EX_Wire[index];

 sourceText = createScrolledText(rowColumn, "text", args, 2,
        sourceData, VoltageSourcesCount, 1);
 XtManageChild (sourceText);
}
if (bFreq) {
 extern int FrequencyCount;
 extern int *FR_IFRQ, *FR_NFRQ;
 extern float  *FR_FMHZ, *FR_DELFRQ;
 int i, j = 0, step;

 for (i = 0; i < FrequencyCount; i++)
  if (FR_IFRQ[i] == 0) {
   step = 0;
   while (step < FR_NFRQ[i]) {
    freqData[j++] = FR_FMHZ[i] + FR_DELFRQ[i] * step;
    step++;
   }
  }
  else {
   step = 0;
   while (step < FR_NFRQ[i]) {
    freqData[j++] = FR_FMHZ[i] * pow((double)FR_DELFRQ[i],(double)step);
    step++;
   }
  }
 freqText = createScrolledText(rowColumn, "text", args, 2,
        freqData, j, 0);
 XtManageChild (freqText);
}
if (bTag) {
 tagText = XmCreateText (rowColumn, "text", args, 2);
 XtManageChild (tagText);
}
if (bAngle) {
 extern int RadiationPatternCount;
 extern int *RP_NPH, *RP_NTH;
 extern float *RP_THETS, *RP_PHIS, *RP_DTH, *RP_DPH;
 int i, j = 0, k = 0, step;

 for (i = 0; i < RadiationPatternCount; i++) {
  step = 0;
  while (step < RP_NTH[i]) {
   thetaData[j++] = RP_THETS[i] + RP_DTH[i] * step;
   step++;
  }
  step = 0;
  while (step < RP_NPH[i]) {
   phiData[k++] = RP_PHIS[i] + RP_DPH[i] * step;
   step++;
  }
 }
 numThetaData = j;
 numPhiData = k;
 angleText = createScrolledText(rowColumn, "text", args, 2,
        phiData, k, 0);
 XtManageChild (angleText);
}
if (bXYZfreq) {
 extern int FrequencyCount, NearElectricCount;
 extern int *FR_IFRQ, *FR_NFRQ, *NE_NRX, *NE_NRY, *NE_NRZ;
 extern float  *FR_FMHZ, *FR_DELFRQ, *NE_XNR, *NE_YNR, *NE_ZNR,
        *NE_DXNR, *NE_DYNR, *NE_DZNR;
 int i, j = 0, x = 0, y = 0, z = 0, step;

 for (i = 0; i < NearElectricCount; i++) {
  step = 0;
  while (step < NE_NRX[i]) {
   xData[x++] = NE_XNR[i] + NE_DXNR[i] * step;
   step++;
  }
  step = 0;
  while (step < NE_NRY[i]) {
   yData[y++] = NE_YNR[i] + NE_DYNR[i] * step;
   step++;
  }
  step = 0;
  while (step < NE_NRZ[i]) {
   zData[z++] = NE_ZNR[i] + NE_DZNR[i] * step;
   step++;
```

```
      }
     }
     numXData = x;
     numYData = y;
     numZData = z;
     for (i = 0; i < FrequencyCount; i++)
      if (FR_IFRQ[i] == 0) {
       step = 0;
       while (step < FR_NFRQ[i]) {
        freqData[j++] = FR_FMHZ[i] + FR_DELFRQ[i] * step;
        step++;
       }
      }
      else {
       step = 0;
       while (step < FR_NFRQ[i]) {
        freqData[j++] = FR_FMHZ[i] * pow((double)FR_DELFRQ[i],(double
)step);
        step++;
       }
      }
     numFreqData = j;

     sel1Text = createScrolledText(rowColumn, "text", args, 2,
           yData, y, 0);
     XtManageChild (sel1Text);
     sel2Text = createScrolledText(rowColumn, "text", args, 2,
           zData, z, 0);
     XtManageChild (sel2Text);
     sel3Text = createScrolledText(rowColumn, "text", args, 2,
           freqData, j, 0);
     XtManageChild (sel3Text);
    }
    if (bXYZfreq2) {
     extern int FrequencyCount;
     extern int *FR_IFRQ, *FR_NFRQ;
     extern float  *FR_FMHZ, *FR_DELFRQ;
     extern int NearMagneticCount;
     extern int *NH_NRX, *NH_NRY, *NH_NRZ;
     extern float  *NH_XNR, *NH_YNR, *NH_ZNR,
           *NH_DXNR, *NH_DYNR, *NH_DZNR;
     int i, j = 0, x = 0, y = 0, z = 0, step;

     for (i = 0; i < NearMagneticCount; i++) {
      step = 0;
      while (step < NH_NRX[i]) {
       xData[x++] = NH_XNR[i] + NH_DXNR[i] * step;
       step++;
      }
      step = 0;
      while (step < NH_NRY[i]) {
       yData[y++] = NH_YNR[i] + NH_DYNR[i] * step;
       step++;
      }
      step = 0;
      while (step < NH_NRZ[i]) {
       zData[z++] = NH_ZNR[i] + NH_DZNR[i] * step;
       step++;
      }
     }
     numXData = x;
     numYData = y;
     numZData = z;
     for (i = 0; i < FrequencyCount; i++)
      if (FR_IFRQ[i] == 0) {
       step = 0;
       while (step < FR_NFRQ[i]) {
        freqData[j++] = FR_FMHZ[i] + FR_DELFRQ[i] * step;
        step++;
       }
      }
      else {
       step = 0;
       while (step < FR_NFRQ[i]) {
        freqData[j++] = FR_FMHZ[i] * pow((double)FR_DELFRQ[i],(double
)step);
        step++;
       }
      }
     numFreqData = j;

     sel1Text = createScrolledText(rowColumn, "text", args, 2,
           yData, y, 0);
     XtManageChild (sel1Text);
     sel2Text = createScrolledText(rowColumn, "text", args, 2,
           zData, z, 0);
     XtManageChild (sel2Text);
     sel3Text = createScrolledText(rowColumn, "text", args, 2,
           freqData, j, 0);
     XtManageChild (sel3Text);
    }

    n = 0;
```

```
  XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNrightAttachment, XmATTACH_WIDGET); n++;
  XtSetArg (args [n], XmNrightWidget, rowColumn); n++;
  XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
  XtSetArg (args [n], XmNtopOffset, 10); n++;
 /**
  XtSetArg (args [n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
  XtSetArg (args [n], XmNbottomWidget, rowColumn); n++;
 **/
  rowColumn = XmCreateRowColumn (form, "rowColumn", args, n);
  XtManageChild (rowColumn);

  n = 0;
  string = XmStringCreateLtoR ("Plot Type:", XmSTRING_DEFAULT_CHARSET);
  XtSetArg (args [n], XmNlabelString, string); n++;
  XtSetArg (args [n], XmNmarginTop, 7); n++;
  XtSetArg (args [n], XmNmarginBottom, 7); n++;
  label = XmCreateLabel (rowColumn, "label", args, n);
  XtManageChild (label);
  XmStringFree (string);

  if (bPower) {
   string = XmStringCreateLtoR ("Antenna Power:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   label = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (label);
   XmStringFree (string);
  }

  if (bSource) {
   string = XmStringCreateLtoR ("Source:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   label = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (label);
   XmStringFree (string);
  }

  if (bFreq) {
   string = XmStringCreateLtoR ("Frequency:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   label = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (label);
   XmStringFree (string);
  }

  if (bTag) {
   string = XmStringCreateLtoR ("Tag:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   label = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (label);
   XmStringFree (string);
  }

  if (bAngle) {
   string = XmStringCreateLtoR ("Phi:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   angleLabel = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (angleLabel);
   XmStringFree (string);
  }

  if ((bXYZfreq)||(bXYZfreq2)) {
   string = XmStringCreateLtoR ("Y:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   sel1Label = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (sel1Label);
   XmStringFree (string);

   string = XmStringCreateLtoR ("Z:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   sel2Label = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (sel2Label);
   XmStringFree (string);

   string = XmStringCreateLtoR ("Frequency:", XmSTRING_DEFAULT_CHARSET);
   XtSetArg (args [0], XmNlabelString, string);
   sel3Label = XmCreateLabel (rowColumn, "label", args, 3);
   XtManageChild (sel3Label);
   XmStringFree (string);
  }

  XtManageChild (form);

  /* Create the action area */
  actionA = createActionArea (pane, actionItems, XtNumber (actionItems));

  XtPopup (plotDialog, XtGrabNone);

  w = w; /* Make compiler happy */

 } /* end openPlotDialog */

 /*****************************************************************
```

```
 * Opens a dialog box to query user for plotting data - plot type,
 * source, frequency, tag & theta.
 */
static void openPlotDialog2 (w, text)
 Widget       w, text;
{
 extern void  runFilter ();

 XtDestroyWidget (XtParent (w));
 switch (optionType) {
  case CURRENTS:
   runFilter (text, NULL, CURRENTS);
   openPlotDialog ((Widget) NULL, "2");
   break;
  case CHARGE:
   runFilter (text, NULL, CHARGE);
   openPlotDialog ((Widget) NULL, "3");
   break;
  case NEAR_ELECTRIC:
   runFilter (text, NULL, NEAR_ELECTRIC);
   openPlotDialog ((Widget) NULL, "5");
                                        break;
  case NEAR_MAGNETIC:
   runFilter (text, NULL, NEAR_MAGNETIC);
                                        openPlotDialog ((Widget) NULL, "6");
  default:
   break;
 }
 w = w; /* Make compiler happy */

} /* end openPlotDialog2 */

/*****************************************************************************
 * Opens a dialog box to query user for plotting data - plot type,
 * source, frequency, tag & theta.
 */
void powerPlotDialog (w, cType)
 Widget       w;
 char         *cType;
{
 optionType = (enum Options) atoi (cType);
 switch (optionType) {
  case CURRENTS:
   createPromptDialog ("Enter antenna input power:",
     "Currents", openPlotDialog2);
   break;
  case CHARGE:
   createPromptDialog ("Enter antenna input power:",
     "Charges", openPlotDialog2);
   break;
  case NEAR_ELECTRIC:
   createPromptDialog ("Enter antenna input power:",
     "Near Electric Fields", openPlotDialog2);
   break;
  case NEAR_MAGNETIC:
   createPromptDialog ("Enter antenna input power:",
     "Near Magnetic Fields", openPlotDialog2);
  default:
   break;
 }
 w = w; /* Make compiler happy */

} /* end powerPlotDialog */

/*****************************************************************************
 * Coupling is the only menuitem in the Output menu which does not call
 * openPlotDialog.  This is because there are no additional inputs
 * required to plot coupling.
 */
void outputMenuCouplingCB (void)
{
 extern void needsPlot ();
 extern Widget topLevel;

 /* Check NEC run status before continuing */
 if (!necRunStatus()) return;

 /* Make sure NEC Output file has been specified */
 if (necOutputFilename == NULL) {
  createMessageDialog (topLevel, "Plot Error",
                                        "NEC Output file must be specified", XmDIALOG_ERROR);
  return;
 }

 needsflt (COUPLING, 0);

 needsPlot ("CP", "");

} /* end outputMenuCouplingCB */

/*****************************************************************************
 * Saves the selected plot type into 'plotType'.
 * Displays the appropriate input labels when the user changes the
```

```
 * plot type for Near Electric Fields or Radiation Patterns.
 */
static void plotTypeCB (w, type)
 Widget      w;
 int         type;
{
 XmString    label1, label2, label3;

 plotType = type;
 if ((optionType == NEAR_ELECTRIC)||(optionType == NEAR_MAGNETIC)) {
  switch (plotType) {
   case 1:   /* Ez Normalized vs. X */
   case 5:
    label1 = XmStringCreateSimple ("Y:");
    label2 = XmStringCreateSimple ("Z:");
    label3 = XmStringCreateSimple ("Frequency:");
    editScrolledText(sel1Text, yData, numYData, 0);
    editScrolledText(sel2Text, zData, numZData, 0);
    editScrolledText(sel3Text, freqData, numFreqData, 0);
    break;
   case 2:
   case 6:
    label1 = XmStringCreateSimple ("X:");
    label2 = XmStringCreateSimple ("Z:");
    label3 = XmStringCreateSimple ("Frequency:");
    editScrolledText(sel1Text, xData, numXData, 0);
    editScrolledText(sel2Text, zData, numZData, 0);
    editScrolledText(sel3Text, freqData, numFreqData, 0);
    break;
   case 3:
   case 7:
    label1 = XmStringCreateSimple ("X:");
    label2 = XmStringCreateSimple ("Y:");
    label3 = XmStringCreateSimple ("Frequency:");
    editScrolledText(sel1Text, xData, numXData, 0);
    editScrolledText(sel2Text, yData, numYData, 0);
    editScrolledText(sel3Text, freqData, numFreqData, 0);
    break;
   case 4:
   case 8:
    label1 = XmStringCreateSimple ("X:");
    label2 = XmStringCreateSimple ("Y:");
    label3 = XmStringCreateSimple ("Z:");
    editScrolledText(sel1Text, xData, numXData, 0);
    editScrolledText(sel2Text, yData, numYData, 0);
    editScrolledText(sel3Text, zData, numZData, 0);
    break;
  }
  XtVaSetValues (sel1Label, XmNlabelString, label1, NULL);
  XtVaSetValues (sel2Label, XmNlabelString, label2, NULL);
  XtVaSetValues (sel3Label, XmNlabelString, label3, NULL);
  XmStringFree (label1);
  XmStringFree (label2);
  XmStringFree (label3);

 } else if (optionType == RADIATION) {
  if ((plotType == 1) || (plotType == 2)) {
   label1 = XmStringCreateSimple ("Phi:");
   editScrolledText(angleText, phiData, numPhiData, 0);
  }
  else {
   label1 = XmStringCreateSimple ("Theta:");
   editScrolledText(angleText, thetaData, numThetaData, 0);
  }
  XtVaSetValues (angleLabel, XmNlabelString, label1, NULL);
  XmStringFree (label1);
 }

 w = w; /* Make compiler happy */

} /* end plotTypeCB */

/*****************************************************************
 * Destroys the widget when cancel button is selected.
 */
static void cancelButtonCB (void)
{
 XtDestroyWidget (plotDialog);

} /* end cancelButtonCB */

/*****************************************************************/
static void plotButtonCB (void)
{
 static char  *codes [] = {"Z", "A", "CR", "Q", "CP", "NE", "NH", "PT"};
 char         inputs [80], *string;
 extern void  needsPlot ();
 extern void  runFilter ();
 int          source, tag;
 float        freq, angle, sel1, sel2, sel3;

 switch (optionType) {
```

```
    case IMPEDANCE:
      string = XmTextGetString (sourceText);
      source = atoi (string);
      XtFree (string);
      sprintf (inputs, "%i %i", plotType, source);
      break;

    case ADMITTANCE:
      string = XmTextGetString (sourceText);
      source = atoi (string);
      XtFree (string);
      sprintf (inputs, "%i %i", plotType, source);
      break;

    case CURRENTS:
      runFilter (powerText, NULL, CURRENTS);
    case CHARGE:
      if (optionType == CHARGE)
        runFilter (powerText, NULL, CHARGE);
      string = XmTextGetString (sourceText);
      source = atoi (string);
      XtFree (string);
      string = XmTextGetString (freqText);
      freq = atof (string);
      XtFree (string);
      string = XmTextGetString (tagText);
      tag = atoi (string);
      XtFree (string);
      sprintf (inputs, "%i %i %f %i", plotType, source, freq, tag);
      break;

    case NEAR_ELECTRIC:
      runFilter (powerText, NULL, NEAR_ELECTRIC);
      string = XmTextGetString (sourceText);
      source = atoi (string);
      XtFree (string);
      string = XmTextGetString (sel1Text);
      sel1 = atof (string);
      XtFree (string);
      string = XmTextGetString (sel2Text);
      sel2 = atof (string);
      XtFree (string);
      string = XmTextGetString (sel3Text);
      sel3 = atof (string);
      XtFree (string);
      sprintf (inputs, "%i %i %f %f %f", plotType, source, sel1, sel2, sel3);
      break;

    case NEAR_MAGNETIC:
      runFilter (powerText, NULL, NEAR_MAGNETIC);
      string = XmTextGetString (sourceText);
      source = atoi (string);
      XtFree (string);
      string = XmTextGetString (sel1Text);
      sel1 = atof (string);
      XtFree (string);
      string = XmTextGetString (sel2Text);
      sel2 = atof (string);
      XtFree (string);
      string = XmTextGetString (sel3Text);
      sel3 = atof (string);
      XtFree (string);
      sprintf (inputs, "%i %i %f %f %f", plotType, source, sel1, sel2, sel3);
      break;

    case RADIATION:
      string = XmTextGetString (sourceText);
      source = atoi (string);
      XtFree (string);
      string = XmTextGetString (freqText);
      freq = atof (string);
      XtFree (string);
      string = XmTextGetString (angleText);
      angle = atof (string);
      XtFree (string);
      sprintf (inputs, "%i %i %f %f", plotType, source, freq, angle);

    default:
      break;
  }

  XtDestroyWidget (plotDialog);
  needsPlot (codes[optionType], inputs);

} /* end plotButtonCB */

/*****************************************************************
 * Set the plotDialog to NULL when window is destroyed.
 */
static void destroyDialogCB (void)
{
  plotDialog = NULL;
```

```
} /* destroyDialogCB */

/*****************************************************************
 * Creates a option menu from the array of data structure MenuItem
 */
static void createOptionMenu (parent, menuItems)
  Widget       parent;
  MenuItem     *menuItems;
{
  Widget       menuPane, widget;
  int          i;
  Arg          args[2];
  XmString     string;

  menuPane = XmCreatePulldownMenu (parent, "menuPane", NULL, 0);

  for (i = 0; menuItems[i].name != NULL; i++) {
    widget = XmCreatePushButton (menuPane, menuItems[i].name, NULL, 0);
    XtAddCallback (widget, XmNactivateCallback, (XtCallbackProc) plotTypeCB,
                   (XtPointer) (i+1));
    XtManageChild (widget);
  }

  string = XmStringCreateSimple ("");
  XtSetArg (args[0], XmNsubMenuId, menuPane);
  XtSetArg (args[1], XmNlabelString, string);
  widget = XmCreateOptionMenu (parent, "menu", args, 2);
  XtManageChild (widget);
  XmStringFree (string);

} /* end createOptionMenu */

/*****************************************************************
 * Returns True if the filter program has already been run for the
 * optiontype.  Otherwise, returns False.
 */
Boolean alreadyFiltered (extension)
  char         *extension;
{
  char         fileName [132], *ptr;
  struct stat  buf, buf2;
  Boolean      filtered;

  /* Construct name of filtered file */
  strcpy(fileName, necOutputFilename);
  ptr = strrchr(fileName,'.');
  if (ptr) *(++ptr) = '\0';
  strcat (fileName, extension);

  /* Get file status */
  if (stat (fileName, &buf) == -1)    /* Probably doesn't exist... */
    return (False);

  /* Return False if time stamp is earlier than necOutputFilename */
  stat (necOutputFilename, &buf2);
  if (buf.st_mtime > buf2.st_mtime)
    filtered = True;
  else
    filtered = False;

  return (filtered);

} /* end alreadyFiltered */


void openPrinterWindow (void)
{
  extern Widget topLevel;

  createMessageDialog (topLevel, "Print Window",
                       "Click on window to be printed.", XmDIALOG_ERROR);

  system
    ("echo \"%1B\" > /dev/tty01; xwd | xpr -device ljet -rv | lpr");

} /* openPrinterWindow */

/*****************************************************************/
void print(w, list)
    Widget w;
    Widget list;
{
  int position;

  XtVaGetValues(list, XmNtopItemPosition, &position, NULL);
  fprintf(stderr, "position = [%d]\n", position);

  w = w; /* Make compiler happy */
}

/*****************************************************************/
void outputTextCB (w, nextText)
    Widget w, nextText;
```

```
{
  extern Widget getTopShell();

  XtSetKeyboardFocus (getTopShell(w), nextText);

  w = w;  /* Make compiler happy */

} /*end nodeCoordTextCB */
```

A.13 fHelp.c:

fHelp.c:

```
/*
 * Filename : fHelp.c
 *
 * Procedures for creating the help window
 */

#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <Xm/MessageB.h>
#include <Xm/RowColumn.h>
#include "control.h"

#ifdef NOT_MOSAIC_HELP

#include <Xm/List.h>
#include <Xm/PanedW.h>
#include <Xm/SelectioB.h>
#include <Xm/Text.h>
#include <X11/Xos.h>        /* for the index() function */

#endif

extern Widget topLevel;

int MosaicPID = 0;  /* Process ID number for Mosaic */

#ifdef NOT_MOSAIC_HELP

static Widget helpText, helpList;
static char helpIndexFile [] =  "toc.txt";
static char helpDataFile [] =   "help.txt";

static void listCB ();

/*****************************************************************************
 * Opens a window for Help
 */
void openHelpWindow (w)
 Widget w;
{
 Widget dialog, pane;
 Arg args [6];
 int textPos = 0;
 FILE *fp;
 char buf [200];
 XmString string;
 extern FILE *efopen ();
 extern void newEscapeAction();

 /* Build the text window */
 dialog = XtVaCreatePopupShell
  (NULL, topLevelShellWidgetClass, XtParent (w),
   XmNtitle,           "User's Manual",
   XmNallowShellResize,  True,
   XmNdeleteResponse,   XmUNMAP,
   NULL);

 newEscapeAction(dialog);

 XtSetArg (args[0], XmNsashIndent, -25);
 XtSetArg (args[1], XmNsashHeight, 15);
 XtSetArg (args[2], XmNsashWidth, 15);
 pane = XmCreatePanedWindow (dialog, "pane", args, 3);
 XtManageChild (pane);

 XtSetArg (args[0], XmNvisibleItemCount, 10);
 helpList = XmCreateScrolledList (pane, "list", args, 1);
 XtAddCallback (helpList, XmNbrowseSelectionCallback, listCB, NULL);
 XtManageChild (helpList);

 XtSetArg (args[0], XmNrows, 20);
 XtSetArg (args[1], XmNcolumns, 80);
 XtSetArg (args[2], XmNeditable, False);
 XtSetArg (args[3], XmNblinkRate, 0);
 XtSetArg (args[4], XmNcursorPositionVisible, False);
 XtSetArg (args[5], XmNeditMode, XmMULTI_LINE_EDIT);
 helpText = XmCreateScrolledText (pane, "text", args, 6);
 XtManageChild (helpText);

 /* Open the help index file */
 if ((fp = efopen (helpIndexFile, "r")) == NULL)
  return;

 /* Read the index data */
```

```
  while ((fgets (buf, 200, fp)) != NULL) {
   /* Remove the carriage return symbol */
   buf [strlen(buf) - 1] = '\0';
   string = XmStringCreateSimple (buf);
   XmListAddItem (helpList, string, 0);
   XmStringFree (string);
  }
  fclose (fp);

  /* Open the help data file */
  if ((fp = efopen (helpDataFile, "r")) == NULL)
   return;

  /* Read the data */
  while ((fgets (buf, 200, fp)) != NULL) {
   XmTextInsert (helpText, textPos, buf);
   textPos += strlen (buf);
  }
  fclose (fp);

  XtPopup (dialog, XtGrabNone);

} /* end openViewWindow */

/*************************************************************************
 * Callback for list widget.  When user selects a list item, a search
 * is made in the text box for it.
 */
static void listCB (w, clientData, cbs)
 Widget w;
 XtPointer clientData;
 XmListCallbackStruct *cbs;
{
 char *string, *searchString, *p;
 int length;
 Boolean found = False;
 XmTextPosition pos;

 /* Get the item selected */
 XmStringGetLtoR (cbs->item, XmSTRING_DEFAULT_CHARSET, &searchString);
 length = strlen (searchString);

 /* Get the help data text */
 string = XmTextGetString (helpText);

 /* Start searching */
 for (p = string; p = index (p, *searchString); p++)
  if (!strncmp (p, searchString, length)) {
   found = True;
   break;
  }

 /* if Found... */
 pos = (XmTextPosition) (p - string);
 XmTextSetTopCharacter (helpText, pos);

 XtFree (searchString);
 XtFree (string);

 /* Make compiler happy */
 w = w;
 clientData = clientData;

} /* end listCB */

#endif

void openAboutWindow ()
{
 Widget messageBox, label;
 XmString okString, xmtitle;
 Arg args [5];
 int n = 0;

 okString = XmStringCreateSimple ("OK");
 xmtitle = XmStringCreateLtoR ("About NEEDS", XmSTRING_DEFAULT_CHARSET);
 XtSetArg (args [n], XmNdialogTitle, xmtitle); n++;
 XtSetArg (args [n], XmNautoUnmanage, False); n++;
 XtSetArg (args [n], XmNcancelLabelString, okString); n++;
 XtSetArg (args [n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
 XtSetArg (args [n], XmNdialogType, XmDIALOG_MESSAGE); n++;
 messageBox = XmCreateMessageDialog (topLevel, "About NEEDS", args, n);

 XtAddCallback (messageBox, XmNcancelCallback,
          (XtCallbackProc) XtDestroyWidget, NULL);
 XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_OK_BUTTON));
 XtUnmanageChild (XmMessageBoxGetChild (messageBox, XmDIALOG_HELP_BUTTON));

 XmStringFree (okString);
 XmStringFree (xmtitle);

 /* Add text describing NEEDS version & responsible parties */
 {
```

```
    Widget rowColumn;
    XmString xmstr;

    rowColumn = XtVaCreateWidget
     ("rowColumn", xmRowColumnWidgetClass, messageBox,
      XmNisAligned, True,
      XmNentryAlignment, XmALIGNMENT_CENTER,
      NULL);

    xmstr = XmStringCreateSimple ("NEEDS");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line1", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple ("(Numerical Electromagnetic");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line2", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple ("Engineering Design System)");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line3", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple (VERSION);
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line4", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple (RELEASE);
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line5", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    label = XmCreateLabel (rowColumn, " ", args, 0);
    XtManageChild (label);

    xmstr = XmStringCreateSimple ("A Product of NAVSEA");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line6", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple ("EMENG Program Manager:  D. R. Cebulski");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line7", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple ("NAVSEA O3K2, (703) 602-7244 x200");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line8", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    label = XmCreateLabel (rowColumn, " ", args, 0);
    XtManageChild (label);

    label = XmCreateLabel (rowColumn, " ", args, 0);
    XtManageChild (label);

    xmstr = XmStringCreateSimple ("Naval Command, Control and Ocean Surveillance Center");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line9", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple ("RDT&E Division");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line10", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    xmstr = XmStringCreateSimple ("San Diego, California 92152-5001");
    XtSetArg (args[0], XmNlabelString, xmstr);
    label = XmCreateLabel (rowColumn, "line11", args, 1);
    XtManageChild (label);
    XmStringFree (xmstr);

    XtManageChild (rowColumn);
  }

  XtManageChild (messageBox);

} /* end openAboutWindow */


void openMosaicWindow ()
```

```
{
  static char command [] = "mosaic -home needs.html";

  if ((MosaicPID = fork ()) == 0)
    execlp ("/bin/sh", "sh", "-c", command, (char *) 0);

} /* end openMosaicWindow */
```

A.14 needs.html, needsin1.html:

needs.html:

```
<H1><B>Numerical Electromagnetic Engineering Design System
(NEEDS)<P>
WORKSTATION ON-LINE HELP</B></H1>
<hr>
The Help Contents lists the available Help topics.  Use the scroll bar to see entries not currently visible in the window.

<H1><B>MENU BAR</B></H1>
The Menu Bar is located at the top of the NEEDS application, just below the Title Bar.  The following describe each of the menu items.  Hot keys for accessing
the specific options are also indicated.<P>

<ul>
<li><A HREF="needfile.html#file"><strong>FILE</strong></A>
<li><B>INPUT</B>
<ul>
<li><A HREF="comments.html#comments">COMMENTS</A>
<li><A HREF="needsin1.html#geometry">GEOMETRY DESCRIPTION</A>
<li><A HREF="needsin2.html#electrical">EDIT CONTROL CARDS</A>
</ul>
<li><B>EXECUTE</B>
<UL>
<LI><A HREF="execute.html#DESCRIPTION">DESCRIPTION SUMMARY - Ctl+D</A>
<LI><A HREF="execute.html#diagnostic">DIAGNOSTICS - Shf+D</A>
<LI><A HREF="execute.html#run">NEC-MOM EXECUTE - Ctl+X</A>
<LI><A HREF="execute.html#status">NEC-MOM RUN STATUS - Shf+X</A>
</UL>
<li><B>RESULT</B></A>
<UL>
<LI><A HREF="result.html#text">TEXT</A>
<LI><A HREF="result.html#plot">PLOT</A>
<LI><A HREF="result.html#visual">VISUALIZATION</A>
</UL>
<li><A HREF="noutput.html#output"><li>OUTPUT</A>
</ul>

<H1><B>MISCELLANEOUS TOPICS</B></H1>
<UL>
<LI><A HREF = "oleo.html#PROCESS">MODELING PROCESS</A>
<LI><A HREF = "oleo.html#DIALOG">DIALOG WINDOWS</A>
<LI><A HREF = "oleo.html#CUSTOM">CUSTOMIZATION</A>
<LI><A HREF = "oleo.html#FILES">NEEDS WORKSTATION FILES</A>
<LI><A HREF = "oleo.html#LIMIT">NEEDS LIMITATIONS</A>
<LI><A HREF = "evaluate.html#eval">EVALUATION OF METHOD OF MOMENTS MODELING</A>
</UL>
```

needsin1.html:

```
<A NAME="geometry"><H3>GEOMETRY DESCRIPTION</H3></a>

The following describe each of the submenu items.  The hot keys for accessing the specific options are also indicated.

<ul>

<li><A HREF="geometry.html#node">NODE COORDINATES - Ctl+N</A>

<li><A HREF="geometry.html#wire">STRAIGHT WIRES - Ctl+W</A>

<li><A HREF="geometry.html#tapered">TAPERED WIRES - Shf+W</A>

<li><A HREF="geometry.html#catenary">CATENARY WIRES  - Ctl+C</A>

<li><A HREF="geometry.html#arc">WIRE ARC - Ctl+A</A>

<li><A HREF="geometry.html#helix">HELIX OR SPIRAL</A>

<li><A HREF="geometry.html#mesh">WIRE MESH SURFACE - Ctl+Z</A>

<li><A HREF="geometry.html#surface">SURFACE PATCHES</A>

<li><A HREF="geometry.html#multiple">MULTIPLE PATCH QUADRANGLE SURFACE </A>
```

```
<li><A HREF="geometry.html#transform">TRANSFORMATIONS - Ctl+T</A>

<li><A HREF="geometry.html#rotate">ROTATIONS - Shf+T</A>

<li><A HREF="geometry.html#reflect">REFLECTIONS - Ctl+R</A>

<li><A HREF="geometry.html#spiral">SPIRAL ORDERING</A>

<li><A HREF="geometry.html#iges">CAD INTERFACE</A>

<li><A HREF="geometry.html#order">EDIT CARD ORDER</A>

</ul>
```

A.15 cVisual.c, fVisual.c:

cVisual.c:

```
/*
 * Callback routines for the Visualization Window
 */

#include "control.h"

extern Widget visualShell;
extern int    necRunStatus ();

void openVisualWindow ()
{
 if (visualShell == NULL) createVisualWindow ();

 XtPopup (visualShell, XtGrabNone);

}
```

fVisual.c:

```
/*
 * Procedures for creating the Visualization Window
 */

#include "control.h"
#include <stdio.h>
#include <stdlib.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/PanedW.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>
#include "actionArea.h"
#include "filter.h"

extern Widget topLevel;
Widget       visualShell = NULL;
int          visualType = 1;

static Widget visualLog, sourceLabel, sourceText, freqLabel, freqText,
  pwrLabel, pwrText;

/* Forward declarations for callbacks */

extern void    cancelButtonCB ();
static void    create3dImage ();
static void    visualButtonCB ();
static void    visualOkButtonCB ();
extern int necRunStatus();

/*****************************************************************************/
void createVisualWindow ()

{
 Widget actionA, pane, form, frame1, rowColumn,
      rowColumn1, label, button;
 Arg args [10];
 int n = 0;
 XmString string;
 Position x, y;
 extern void onlyDigitsCB ();
 static ActionAreaItem actionItems[] = {
    {"Ok", visualOkButtonCB, NULL},
    {"Cancel", cancelButtonCB, NULL},
 };
 extern void newEscapeAction();
 extern Widget createScrolledText();
 extern int sourceData[];
 extern float freqData[];
 extern int numFreqData, VoltageSourcesCount, FrequencyCount;
 extern int *FR_IFRQ, *FR_NFRQ;
 extern float  *FR_FMHZ, *FR_DELFRQ;
 int i, j = 0, step;
 extern int *EX_Wire;

 XtTranslateCoords (topLevel, (Position) 0, (Position) 0, &x, &y);
 XtSetArg (args [n], XmNx, x); n++;
 XtSetArg (args [n], XmNy, y + 100); n++;
 visualShell =
   XtCreatePopupShell ("Visualization", topLevelShellWidgetClass,
               topLevel, args, n);
```

```
newEscapeAction(visualShell);

XtSetArg (args[n], XmNsashWidth, 1); n++;
XtSetArg (args[n], XmNsashHeight, 1); n++;
pane = XmCreatePanedWindow (visualShell, "pane", args, n);
XtManageChild (pane);

/* Create control area */
form = XmCreateForm (pane, "form", NULL, 0);

string = XmStringCreateSimple ("Select data type to be displayed:");
XtSetArg (args[n], XmNlabelString, string); n++;
XtSetArg (args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNleftOffset, 15); n++;
XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNtopOffset, 15); n++;
label = XmCreateLabel (form, "label", args, n);
XtManageChild (label);
XmStringFree (string);

/* Create frame to hold data type options */
n = 0;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg (args [n], XmNtopWidget, label); n++;
XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNleftOffset, 10); n++;
XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNrightOffset, 10); n++;
frame1 = XmCreateFrame (form, "frame", args, n);

/* Create the row column box */
n = 0;
XtSetArg (args [n], XmNpacking, XmPACK_COLUMN); n++;
XtSetArg (args [n], XmNnumColumns, 12); n++;
XtSetArg (args [n], XmNradioBehavior, True); n++;
XtSetArg (args [n], XmNorientation, XmHORIZONTAL); n++;
XtSetArg (args [n], XmNisHomogeneous, FALSE); n++;
rowColumn = XmCreateRowColumn (frame1, "rowColumn", args, n);

/* Create the Geometry data type buttons */
XtVaCreateManagedWidget ("GEOMETRY", xmLabelWidgetClass, rowColumn, NULL);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);
XtSetArg (args [0], XmNset, True);
button = XmCreateToggleButton (rowColumn, "Segmentation", args, 1);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 1);
button = XmCreateToggleButton (rowColumn, "Wire Radius", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 2);
button = XmCreateToggleButton (rowColumn, "Segment to Radius Ratio",
                args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 3);
button = XmCreateToggleButton (rowColumn, "Wire Connections", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 4);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);

/* Create Charge & Currents data type buttons */
XtVaCreateManagedWidget ("CURRENTS & CHARGES", xmLabelWidgetClass,
            rowColumn, NULL);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);
button = XmCreateToggleButton (rowColumn, "Current Magnitude", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 8);
button = XmCreateToggleButton (rowColumn, "Current Phase", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 10);
button = XmCreateToggleButton (rowColumn, "Charge", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 11);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);

/* Create Near & Far Field data type buttons */
XtVaCreateManagedWidget ("FIELDS", xmLabelWidgetClass, rowColumn, NULL);
XtVaCreateManagedWidget (" ", xmLabelWidgetClass, rowColumn, NULL);
button = XmCreateToggleButton (rowColumn, "Z-component of E-normal",
                args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 13);
button = XmCreateToggleButton (rowColumn, "Total E-normal", args, 0);
```

```
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 14);
button = XmCreateToggleButton (rowColumn, "X-component of H-normal",
                args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 15);
button = XmCreateToggleButton (rowColumn, "Y-component of H-normal",
                args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 16);
button = XmCreateToggleButton (rowColumn, "E-theta", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 17);
button = XmCreateToggleButton (rowColumn, "E-phi", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNvalueChangedCallback, visualButtonCB,
        (XtPointer) 19);

XtManageChild (rowColumn);
XtManageChild (frame1);

/* Create the row column box */
n = 0;
XtSetArg (args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNleftOffset, 10); n++;
XtSetArg (args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg (args[n], XmNtopWidget, frame1); n++;
XtSetArg (args[n], XmNtopOffset, 10); n++;
XtSetArg (args[n], XmNorientation, XmHORIZONTAL); n++;
rowColumn1 = XmCreateRowColumn (form, "rowColumn", args, n);
XtManageChild (rowColumn1);

n = 0;
string = XmStringCreateSimple ("Select Log or Linear display:");
XtSetArg (args[n], XmNlabelString, string); n++;
label = XmCreateLabel (rowColumn1, "label", args, n);
XtManageChild (label);
XmStringFree (string);

/* Create the row column box */
n = 0;
XtSetArg (args [n], XmNpacking, XmPACK_COLUMN); n++;
XtSetArg (args [n], XmNorientation, XmHORIZONTAL); n++;
XtSetArg (args [n], XmNradioBehavior, True); n++;
rowColumn = XmCreateRowColumn (rowColumn1, "rowColumn", args, n);

visualLog = XmCreateToggleButton (rowColumn, "Log", args, 0);
XtManageChild (visualLog);
button = XtVaCreateManagedWidget ("Linear", xmToggleButtonWidgetClass,
                rowColumn, XmNset, True, NULL);
XtManageChild (rowColumn);
/*
XtManageChild (frame);
*/

/* Create RowColumn box for Source & Frequency specification */
n = 0;
XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNleftOffset, 10); n++;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg (args [n], XmNtopWidget, rowColumn1); n++;
XtSetArg (args [n], XmNtopOffset, 10); n++;
XtSetArg (args [n], XmNorientation, XmVERTICAL); n++;
XtSetArg (args [n], XmNnumColumns, 4); n++;
XtSetArg (args [n], XmNpacking, XmPACK_COLUMN); n++;
rowColumn = XmCreateRowColumn (form, "rowColumn", args, n);
XtManageChild (rowColumn);

/* Create Source label */
n = 0;
string = XmStringCreateLtoR ("Source:", XmSTRING_DEFAULT_CHARSET);
XtSetArg (args [n], XmNlabelString, string); n++;
XtSetArg (args [n], XmNmappedWhenManaged, False); n++;
sourceLabel = XmCreateLabel (rowColumn, "sourceLabel", args, n);
XtManageChild (sourceLabel);
XmStringFree (string);

/* Create Antenna Power label */
n = 0;
string = XmStringCreateLtoR ("Antenna Power:", XmSTRING_DEFAULT_CHARSET);
XtSetArg (args [n], XmNlabelString, string); n++;
XtSetArg (args [n], XmNmappedWhenManaged, False); n++;
pwrLabel = XmCreateLabel (rowColumn, "pwrLabel", args, n);
XtManageChild (pwrLabel);
XmStringFree (string);

/* Create Source text */
n = 0;
XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
```

```
    XtSetArg (args [n], XmNcolumns, 11); n++;
    XtSetArg (args [n], XmNmappedWhenManaged, False); n++;
    for (i = 0; i < VoltageSourcesCount; i++)
      sourceData[i] = EX_Wire[i];

    sourceText = createScrolledText(rowColumn, "sourceText", args, n,
            sourceData, VoltageSourcesCount, 1);

    XtAddCallback (sourceText, XmNmodifyVerifyCallback, onlyDigitsCB, NULL);
    XtManageChild (sourceText);

    /* Create Antenna Power text */
    n = 0;
    XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
    XtSetArg (args [n], XmNcolumns, 11); n++;
    XtSetArg (args [n], XmNmappedWhenManaged, False); n++;
    pwrText = XmCreateText (rowColumn, "pwrText", args, n);
    XtManageChild (pwrText);

    XmTextSetString(pwrText, "1000");

    /* Create Frequency label */
    n = 0;
    string = XmStringCreateLtoR ("Frequency:", XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args [n], XmNlabelString, string); n++;
    XtSetArg (args [n], XmNmappedWhenManaged, False); n++;
    freqLabel = XmCreateLabel (rowColumn, "freqLabel", args, n);
    XtManageChild (freqLabel);
    XmStringFree (string);

    /* Dummy filler */
    XtVaCreateManagedWidget ("     ", xmLabelWidgetClass, rowColumn, NULL);

    /* Create Frequency text */
    n = 0;
    XtSetArg (args [n], XmNeditMode, XmSINGLE_LINE_EDIT); n++;
    XtSetArg (args [n], XmNcolumns, 11); n++;
    XtSetArg (args [n], XmNmappedWhenManaged, False); n++;
    for (i = 0; i < FrequencyCount; i++)
     if (FR_IFRQ[i] == 0) {
        step = 0;
        while (step < FR_NFRQ[i]) {
          freqData[j++] = FR_FMHZ[i] + FR_DELFRQ[i] * step;
          step++;
        }
      }
      else {
        step = 0;
        while (step < FR_NFRQ[i]) {
          freqData[j++] = FR_FMHZ[i] * pow((double)FR_DELFRQ[i],(double)step);
          step++;
        }
      }
    numFreqData = j;
    freqText = createScrolledText(rowColumn, "freqText", args, n,
            freqData, j, 0);

    XtManageChild (freqText);

    /* Create the action area */
    actionA = createActionArea (pane, actionItems, XtNumber (actionItems));

    XtManageChild (form);

} /* end createVisualWindow */

static void visualButtonCB (w, type, state)
  Widget w;
  int type;
  XmToggleButtonCallbackStruct *state;
{
  if (type > 4) {
    XtMapWidget (sourceLabel);
    XtMapWidget (sourceText);
    XtMapWidget (freqLabel);
    XtMapWidget (freqText);
    if ((type >= 8) && (type <= 16)) {
      XtMapWidget (pwrLabel);
      XtMapWidget (pwrText);
    } else {
      XtUnmapWidget (pwrLabel);
      XtUnmapWidget (pwrText);
    }
  } else {
    XtUnmapWidget (sourceLabel);
    XtUnmapWidget (sourceText);
    XtUnmapWidget (freqLabel);
    XtUnmapWidget (freqText);
    XtUnmapWidget (pwrLabel);
    XtUnmapWidget (pwrText);
  }

  if (state->set)
```

```
    visualType = type;
  else
    visualType = 0;

  w = w;  /* Make compiler happy */

} /* end visualButtonCB */


static void visualOkButtonCB (void)
{
  Boolean logFlag;
  int type, source;
  float freq;

  /* Get selected visualization output */
      type = visualType;
  logFlag = XmToggleButtonGetState (visualLog);
  if (type > 7) {
                              if (!necRunStatus()) return;
  }

  if ((visualType == 8) || (visualType == 11) || (visualType > 16)) {
    if (logFlag)
      type++;
  } else if (visualType < 4) {
    if (logFlag)
      type += 4;
  }

  /* Get current values for source & frequency if needed */
  if (type > 7) {
    source = atoi (XmTextGetString (sourceText));
    freq = atof (XmTextGetString (freqText));
  } else {
    source = 0;
    freq = 0;
  }

  XtPopdown (visualShell);

  create3dImage (type, source, freq);

} /* visualOkButtonCB */


static void create3dImage (type, source, freq)
  int type, source;
  float freq;
{
  extern Boolean alreadyFiltered ();
  extern void createMessageDialog ();
  extern void geometryFilter ();
  extern void necDisplay ();
  extern char *necInputFilename, *necOutputFilename;
  extern Widget topLevel;

  if (type < 17)
    /* Run geometryFilter to calculate data needed for geomtry */
    geometryFilter ();

  if (type > 7) {

    /* Make sure NEC Output file has been specified */
    if (necOutputFilename == NULL) {
      createMessageDialog (topLevel, "Visualization Error",
                  "NEC Output file must be specified",
                  XmDIALOG_ERROR);
      return;
    }
    /* Make sure .rcr file exists */
    if ((type >= 8) && (type <= 10))
      runFilter (pwrText, NULL, CURRENTS);
    else if ((type == 11) || (type == 12))
      runFilter (pwrText, NULL, CHARGE);
    else if ((type == 13) || (type == 14))
      runFilter (pwrText, NULL, NEAR_ELECTRIC);
    else if ((type == 15) || (type == 16))
      runFilter (pwrText, NULL, NEAR_MAGNETIC);
    else if (!alreadyFiltered (fileExts[RADIATION])) {
      needsfilt (RADIATION, 0);
    }
  }

    necDisplay (necInputFilename, type, source, freq);

} /* end create3dImage */
```

A.16 geofilt.c:

geofilt.c:

```
/************** GEOFILT.C ********************************************
 * Program provides filtered output files from the input of NEC-MoM
 * Original program was written in Fortran by Linda Russell.
 * This program was ported to C by Darlene Wentworth.
 * 6/12/95 - wire connection routine speeded up by Linda Russell
 *********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <X11/Intrinsic.h>
#include "filter.h"
#include "cFileMenu.h"
#include "control.h"

#define min(a,b)        (a>b ? b : a)
#define max(a,b)        (a<b ? b : a)

extern int EnvIndex;

extern int DimIndex;
extern float DimensionsScale [];

int NumWires, NumSegs;
float Xmin, Xmax, Ymin, Ymax, Zmin, Zmax, Xshift, Yshift, Zshift, Scale,
  SegMin, SegMax, RadMin, RadMax, SrMin, SrMax;
float *Xnode1, *Ynode1, *Znode1, *Xnode2, *Ynode2, *Znode2;
float *Segs, *Rads, *SrRatio;

/* Indexes into global arrays */
int *Jseg, *Jwire, *Iseg, *Irad, *Isr, *Icon, *Iseglg, *Iradlg, *Isrlg;

void geometryFilter ()
{
 float gscale;
 float cdr, bRadius, eRadius, wLength, sLength, sRadius,
  xDif, yDif, zDif, difMax;
 int i, j, nSegs, numNodes, inode, jnode;
 int inode1, inode2, jnode1, jnode2;
 int total_wires;
 float *gc_rad1;
 float *gc_rad2;
 Boolean *connect;

 /* Initial values */
 gscale = DimensionsScale[DimIndex];
 cdr = acos (0.0) / 90.0;
 NumSegs = 0;
 numNodes = 0;
 NumWires = 0;

 /* Calculate total number of segments and allocate arrays */
 nSegs = 0;
 for (i = 0; i < SWireCount; i++)
   nSegs += GW_NS[i];
 for (i = 0; i < TaperWireCount; i++)
   nSegs += GC_NS[i];
 total_wires = SWireCount + TaperWireCount;
 Jseg = (int *) XtMalloc (sizeof (int) * nSegs);
 Jwire = (int *) XtMalloc (sizeof (int) * nSegs);
 Iseg = (int *) XtMalloc (sizeof (int) * nSegs);
 Irad = (int *) XtMalloc (sizeof (int) * nSegs);
 Isr = (int *) XtMalloc (sizeof (int) * nSegs);
 Icon = (int *) XtMalloc (sizeof (int) * nSegs);
 Iseglg = (int *) XtMalloc (sizeof (int) * nSegs);
 Iradlg = (int *) XtMalloc (sizeof (int) * nSegs);
 Isrlg = (int *) XtMalloc (sizeof (int) * nSegs);
 Segs = (float *) XtMalloc (sizeof (float) * nSegs);
 Rads = (float *) XtMalloc (sizeof (float) * nSegs);
 SrRatio = (float *) XtMalloc (sizeof (float) * nSegs);
 Xnode1 = (float *) XtMalloc (sizeof (float) * nSegs);
 Ynode1 = (float *) XtMalloc (sizeof (float) * nSegs);
 Znode1 = (float *) XtMalloc (sizeof (float) * nSegs);
 Xnode2 = (float *) XtMalloc (sizeof (float) * nSegs);
 Ynode2 = (float *) XtMalloc (sizeof (float) * nSegs);
 Znode2 = (float *) XtMalloc (sizeof (float) * nSegs);
 connect = (Boolean *) XtMalloc (sizeof (Boolean) * (total_wires * 2 + 1));
 if (TaperWireCount > 0) {
   gc_rad1 = (float *) XtMalloc (sizeof (float) * TaperWireCount);
   gc_rad2 = (float *) XtMalloc (sizeof (float) * TaperWireCount);
 }

 /* Process straight wires to find minimum x, y, and z */
 for (i = 0; i < SWireCount; i++) {
   float x1, y1, z1, x2, y2, z2;
```

```
int ix;

NumWires++;
connect[(NumWires-1)*2] = False;
connect[(NumWires-1)*2+1] = False;
x1 = X[GW_END1[i] - 1]*gscale;
y1 = Y[GW_END1[i] - 1]*gscale;
z1 = Z[GW_END1[i] - 1]*gscale;
x2 = X[GW_END2[i] - 1]*gscale;
y2 = Y[GW_END2[i] - 1]*gscale;
z2 = Z[GW_END2[i] - 1]*gscale;
ix = GW_NS[i];
if (i == 0) {
  Xmin = Xmax = x1;
  Ymin = Ymax = y1;
  Zmin = Zmax = z1;
  SegMin = 10000.0;
  RadMin = 10000.0;
  SrMin = 10000.0;
  SegMax = 0.0;
  RadMax = 0.0;
  SrMax = 0.0;
}
Xmin = min (Xmin, x1);
Ymin = min (Ymin, y1);
Zmin = min (Zmin, z1);
Xmax = max (Xmax, x1);
Ymax = max (Ymax, y1);
Zmax = max (Zmax, z1);
Xmin = min (Xmin, x2);
Ymin = min (Ymin, y2);
Zmin = min (Zmin, z2);
Xmax = max (Xmax, x2);
Ymax = max (Ymax, y2);
Zmax = max (Zmax, z2);

/* Straight Wire info */
if (GW_RAD[i] >= 0.000001) {
  for (j = 0; j < GW_NS[i]; j++) {
    Jwire[NumSegs] = NumWires;
    Jseg[NumSegs] = j + 1;
    Xnode1[NumSegs] = x1 + j * (x2 - x1)/ix;
    Ynode1[NumSegs] = y1 + j * (y2 - y1)/ix;
    Znode1[NumSegs] = z1 + j * (z2 - z1)/ix;
    Xnode2[NumSegs] = x1 + (j + 1) * (x2 - x1)/ix;
    Ynode2[NumSegs] = y1 + (j + 1) * (y2 - y1)/ix;
    Znode2[NumSegs] = z1 + (j + 1)  * (z2 - z1)/ix;
    Segs[NumSegs] = sqrt (pow ((x1 - x2), 2) + pow ((y1 -y2), 2) +
                    pow ((z1 - z2), 2)) / ix;
    Rads[NumSegs] = GW_RAD[i]*gscale;
    SrRatio[NumSegs] = Segs[NumSegs] / Rads[NumSegs];
    SegMin = min (SegMin, Segs[NumSegs]);
    RadMin = min (RadMin, Rads[NumSegs]);
    SrMin = min (SrMin, SrRatio[NumSegs]);
    SegMax = max (SegMax, Segs[NumSegs]);
    RadMax = max (RadMax, Rads[NumSegs]);
    SrMax = max (SrMax, SrRatio[NumSegs]);
    NumSegs++;
    numNodes += 2;
  }
 }
}

/* Process tapered wires to find minimum x, y, and z */
for (i = 0; i < TaperWireCount; i++) {
  float x1, y1, z1, x2, y2, z2;
  int ix;

  NumWires++;
  connect[(NumWires-1)*2] = False;
  connect[(NumWires-1)*2+1] = False;
  x1 = X[GC_END1[i] - 1]*gscale;
  y1 = Y[GC_END1[i] - 1]*gscale;
  z1 = Z[GC_END1[i] - 1]*gscale;
  x2 = X[GC_END2[i] - 1]*gscale;
  y2 = Y[GC_END2[i] - 1]*gscale;
  z2 = Z[GC_END2[i] - 1]*gscale;
  ix = GC_NS[i];
                          bRadius = GC_RAD1[i]*gscale;
                          eRadius = GC_RAD2[i]*gscale;
  if (i == 0) {
    Xmin = Xmax = x1;
    Ymin = Ymax = y1;
    Zmin = Zmax = z1;
    SegMin = 10000.0;
    RadMin = 10000.0;
    SrMin = 10000.0;
    SegMax = 0.0;
    RadMax = 0.0;
    SrMax = 0.0;
  }
  Xmin = min (Xmin, x1);
  Ymin = min (Ymin, y1);
```

```
    Zmin = min (Zmin, z1);
    Xmax = max (Xmax, x1);
    Ymax = max (Ymax, y1);
    Zmax = max (Zmax, z1);
    Xmin = min (Xmin, x2);
    Ymin = min (Ymin, y2);
    Zmin = min (Zmin, z2);
    Xmax = max (Xmax, x2);
    Ymax = max (Ymax, y2);
    Zmax = max (Zmax, z2);

    /* For each segment, find nodes, lengths, etc. */
    for (j = 0; j < GC_NS[i]; j++) {
     Jwire[NumSegs] = NumWires;
     Jseg[NumSegs] = j + 1;
     if (j > 0) {
      x1 = Xnode2[NumSegs - 1];
      y1 = Ynode2[NumSegs - 1];
      z1 = Znode2[NumSegs - 1];
     }
     wLength = sqrt (pow ((x1 - x2), 2) + pow ((y1 -y2), 2) +
                pow ((z1 - z2), 2));
     if (j == 0) {
      if (abs (1 - GC_RDEL[i]) > 0.0001)
       sLength = wLength * (1 - GC_RDEL[i]) / (1 - pow (GC_RDEL[i], ix));
      else
       sLength = wLength / ix;
     } else if (j == ix)
      sLength = wLength;
     else
      sLength = GC_RDEL[i] * sLength;
     sRadius = bRadius + j * (eRadius - bRadius)/(ix - 1);
     if (j==0) gc_rad1[i] = sRadius;
     if (j==GC_NS[i]-1) gc_rad2[i] = sRadius;
     Xnode1[NumSegs] = x1;
     Ynode1[NumSegs] = y1;
     Znode1[NumSegs] = z1;
     Xnode2[NumSegs] = x1 + (sLength / wLength) * (x2 - x1);
     Ynode2[NumSegs] = y1 + (sLength / wLength) * (y2 - y1);
     Znode2[NumSegs] = z1 + (sLength / wLength) * (z2 - z1);
     Segs[NumSegs] = sLength;
     Rads[NumSegs] = sRadius;
     SrRatio[NumSegs] = sLength / sRadius;
     SegMin = min (SegMin, Segs[NumSegs]);
     RadMin = min (RadMin, Rads[NumSegs]);
     SrMin = min (SrMin, SrRatio[NumSegs]);
     SegMax = max (SegMax, Segs[NumSegs]);
     RadMax = max (RadMax, Rads[NumSegs]);
     SrMax = max (SrMax, SrRatio[NumSegs]);
     NumSegs++;
     numNodes += 2;
    }
   }

   /* See which wires are connected */
   for (i = 0; i < NumWires; i++) {
    inode = i * 2;
    if (i < SWireCount) {
    /* Straight Wire */
     inode1 = GW_END1[i];
     inode2 = GW_END2[i];
     if (EnvIndex != FREE_SPACE) {
      if (fabs(Z[inode1 - 1]) <= GW_RAD[i]) connect[inode] = True;
      if (fabs(Z[inode2 - 1]) <= GW_RAD[i]) connect[inode + 1] = True;
     }
/*
     if (Z[inode1 - 1] <= GW_RAD[i]) connect[inode] = True;
     if (Z[inode2 - 1] <= GW_RAD[i]) connect[inode + 1] = True;
*/
    } else {
    /* Tapered Wire */
     inode1 = GC_END1[i - SWireCount];
     inode2 = GC_END2[i - SWireCount];
     if (EnvIndex != FREE_SPACE) {
      if (fabs(Z[inode1 - 1]) <= gc_rad1[i]) connect[inode] = True;
      if (fabs(Z[inode2 - 1]) <= gc_rad2[i]) connect[inode+1] = True;
     }
/*
     if (Z[inode1 - 1] <= gc_rad1[i]) connect[inode] = True;
     if (Z[inode2 - 1] <= gc_rad2[i]) connect[inode+1] = True;
*/
    }
    for (j = i+1; j < NumWires; j++) {
     jnode = j*2;
     if (j < SWireCount) {
     /* Straight Wire */
      jnode1 = GW_END1[j];
      jnode2 = GW_END2[j];
     } else {
     /* Tapered Wire */
      jnode1 = GC_END1[j - SWireCount];
      jnode2 = GC_END2[j - SWireCount];
     }
```

```
      if (inode1 == jnode1) {
       connect[inode] = True;
       connect[jnode] = True;
      }
      if (inode1 == jnode2) {
       connect[inode] = True;
       connect[jnode + 1] = True;
      }
      if (inode2 == jnode1) {
       connect[inode + 1] = True;
       connect[jnode] = True;
      }
      if (inode2 == jnode2) {
       connect[inode + 1] = True;
       connect[jnode + 1] = True;
      }
     }
    }

    /* Calculate shifts */
    Xshift = (Xmax + Xmin) / 2.0;
    Yshift = (Ymax + Ymin) / 2.0;
    Zshift = Zmin;

    /* Calculate Scale factor */
    xDif = Xmax - Xmin;
    yDif = Ymax - Ymin;
    zDif = Zmax - Zmin;
    difMax = xDif;
    difMax = max (yDif, difMax);
    difMax = max (zDif, difMax);
    Scale = 4.0 / difMax;

    for (i = 0; i < NumSegs; i++) {
     inode = 2 * (Jwire[i] - 1);
     if (SegMax == SegMin) {
      Iseg[i] = 0;
      Iseglg[i] = 0;
     } else {
      Iseg[i] = (Segs[i] - SegMin) / (SegMax - SegMin) * 7;
      Iseglg[i] = (log10 (SegMin/Segs[i]) / log10 (SegMin/SegMax)) * 7;
      if (Segs[i] == SegMax) {
       Iseg[i] = 6;
       Iseglg[i] = 6;
      }
     }
     if (RadMax == RadMin) {
      Irad[i] = 0;
      Iradlg[i] = 0;
     } else {
      Irad[i] = (Rads[i] - RadMin) / (RadMax - RadMin) * 7;
      Iradlg[i] = (log10 (RadMin/Rads[i]) / log10 (RadMin/RadMax)) * 7;
      if (Rads[i] == RadMax) {
       Irad[i] = 6;
       Iradlg[i] = 6;
      }
     }
     if (SrMax == SrMin) {
      Isr[i] = 0;
      Isrlg[i] = 0;
     } else {
      Isr[i] = (SrRatio[i] - SrMin) / (SrMax - SrMin) * 7;
      Isrlg[i] = (log10 (SrMin/SrRatio[i]) / log10 (SrMin/SrMax)) * 7;
      if (SrRatio[i] == SrMax) {
       Isr[i] = 6;
       Isrlg[i] = 6;
      }
     }
     if (connect[inode] && connect[inode+1]) Icon[i] = 0;
     if (connect[inode] && !connect[inode+1]) Icon[i] = 2;
     if (!connect[inode] && connect[inode+1]) Icon[i] = 2;
     if (!connect[inode] && !connect[inode+1]) Icon[i] = 4;
    }

    XtFree (connect);
    connect = NULL;
    if (TaperWireCount > 0) {
     XtFree((char *) gc_rad1);
     gc_rad1 = NULL;
     XtFree((char *) gc_rad2);
     gc_rad2 = NULL;
    }

    /* Output data for testing...
     printf ("\t%d\t%d\n", NumWires, NumSegs);
     printf ("%f\t%f\n", Xmin, Xmax);
     printf ("%f\t%f\n", Ymin, Ymax);
     printf ("%f\t%f\n", Zmin, Zmax);
     printf ("%f\t%f\n", Xshift, Yshift);
     printf ("%f\t%f\n", Zshift, Scale);
     printf ("%f\t%f\n", SegMin, SegMax);
     printf ("%f\t%f\n", RadMin, RadMax);
     printf ("%f\t%f\n\n", SrMin, SrMax);
```

```
  for (i = 0; i < NumSegs; i++) {
    printf ("%5d%5d%3d%3d%3d%3d\n", Jseg[i], Jwire[i], Iseg[i], Irad[i],
          Isr[i], Icon[i]);
    printf ("%3d%3d%3d\n", Iseglg[i], Iradlg[i], Isrlg[i]);
    printf ("%10.4f%10.4f%10.4f\n", Segs[i], Rads[i], SrRatio[i]);
    printf ("%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n",
          Xnode1[i], Ynode1[i], Znode1[i], Xnode2[i], Ynode2[i], Znode2[i]);
  }
 */

} /* end geometryFilter */
```

A.17 needsflt.c:

needsflt.c:

```
/************** NECFILT.C ***********************************************
 * Program provides filtered output files from the output of NEC-MoM
 * Original program was a series of program developed by Lance Koyama,
 * NRaD Code 82.
 * This program was ported to C by Darlene Wentworth.
 ************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <Xm/DialogS.h>
#include "filter.h"

extern FILE *efopen ();

static void couple ();
static void current ();
static void charge ();
static void electric ();
static void extract ();
static void impedance ();
static void magnetic ();
static void nearField ();
static void pattern ();

static FILE *inFilePtr;

/************************************************************************
 * Checks if NEC execution completed successfully.  If not, displays
 * error message.  Returns 1, if successful.  Otherwise, returns 0;
 ************************************************************************/
int necRunStatus (void)
{
 char msg[132], command[132];
 int val;
 extern char *necOutputFilename;
 extern Widget topLevel;

 /* Make sure that NEC execution was successful */
 sprintf (command, "tail %s | grep \"RUN TIME\" > /dev/null",
       necOutputFilename);
 val = system(command);
 if (val) {
  sprintf (msg, "[%s] - NEC execution was unsuccessful.",
        necOutputFilename);
  createMessageDialog(topLevel, "NEC execution", msg, XmDIALOG_ERROR);
  return (0);
 } else
  return (1);

} /* end necRunStatus */


/************************************************************************
 * Filters the NEC output file for specified products.
 ************************************************************************/

void needsflt (ftype, normal)
    int ftype;          /* Type of filtering */
    float normal;       /* Power */
{
 char infile [132], *ptr;
 extern char *necOutputFilename;

 /* determine and open input file */
 strcpy(infile, necOutputFilename);
 ptr = strrchr(infile, (int)'.');
 if (ptr) *(ptr) = '\0';

 /* Open the NEC output file */
 if ((inFilePtr = efopen (necOutputFilename, "r")) == NULL)
  return;

 switch (ftype) {
 case IMPEDANCE:
  impedance (infile);
  break;
 case ADMITTANCE:
  impedance (infile);
  break;
 case CURRENTS:
  current (infile, normal);
  break;
 case CHARGE:
  charge (infile, normal);
  break;
```

```
  case COUPLING:
    couple (infile);
    break;
  case NEAR_ELECTRIC:
    electric (infile, normal);
    break;
  case NEAR_MAGNETIC:
    magnetic (infile, normal);
    break;
  case RADIATION:
    pattern (infile);
    break;
  default:
    break;
  }

  printf ("\nNEC filtered file is available in ***.%s\n",
          fileExts[ftype]);
  fclose (inFilePtr);

} /* end necFilter */

/*****************************************************************************
 * couple
 *
 * Description: Pulls out coupling information from the NEC output file.
 *
 *****************************************************************************/

static void couple (infile)
    char *infile;
{
  char outfile [132], line [132];
  float runtime, freq, zre, zim, power, cpl;
  int i, nseg, npatch, iextag;
  static char ext [] = ".rcp";
  FILE *fp;

  /* Create output filename & open file */
  strcpy(outfile, infile);
  strcat(outfile, ext);
  if ((fp = efopen (outfile, "w")) == NULL)
    return;

  /* Start file read and write */
  runtime = 0;
  npatch = 0;
  while ((fgets (line, MAXLINE, inFilePtr)) != NULL) {

    /* Find total segments used */
    if (strstr (line, "TOTAL SEGMENTS USED") != NULL) {
      sscanf (line, " TOTAL SEGMENTS USED=%d", &nseg);
      printf ("%.72s\n", line);
      fgets (line, MAXLINE, inFilePtr);
      npatch = 0;
      if (strstr (line, "PATCH") != NULL) {
        sscanf (line, " TOTAL PATCHES USED=%d", &npatch);
        printf ("%.72s\n", line);
      }
      fprintf (fp, "\n \"%s - %3d segments, %4d patches\"\n",
               infile, nseg, npatch);
      fprintf (fp, "%10s%14s\n", "FREQUENCY", "ISOLATION");
      fprintf (fp, "%8s%13s\n", "(MHz)", "(dB)");

    /* Skip over segmentation data */
    } else if (strstr (line, "- SEGMENTATION DATA -") != NULL) {
      for (i = 0; i < nseg; i++) fgets (line, MAXLINE, inFilePtr);

    /* Print frequencies to stdout */
    } else if (strstr (line, "- FREQUENCY -") != NULL) {
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      sscanf (line, " FREQUENCY=%f MHZ", &freq);
      printf ("%3s%8.2f%4s\n", "FR", freq, "MHz");

    /* Get antenna input parameters */
    } else if (strstr (line, "- ANTENNA INPUT PARAMETERS -") != NULL) {
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      sscanf (line, "%d %*d %*f %*f %*f %*f %f %f %*f %*f %f",
              &iextag, &zre, &zim, &power);

    /* Write out frequencies & coupling to .rcp file */
    } else if (strstr (line, "- ISOLATION DATA -") != NULL) {
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      sscanf (line, "%*d %*d %*d %*d %*d %*d %f", &cpl);
      fprintf (fp, "%8.2f%14.3f\n", freq, -cpl);
```

```
  }
 }
 fclose (fp);
} /* end couple */

/*****************************************************************************
 * current
 *
 * Description: Pulls out current information from the NEC output file.
 *
 *****************************************************************************/

static void current (infile, normal)
   char *infile;
   float normal;
{
 char outfile [132], line [132];
 float runtime, freq, power, zre, zim, x, y, z, curM, curP, scale,
     curNor;
 int i, nseg, ncur, npatch, kfreq, iextag, idum, iseg, itag;
 static char ext [] = ".rcr";
 FILE *fp;

 /* Create output filename & open file */
 strcpy(outfile, infile);
 strcat(outfile, ext);
 if ((fp = efopen (outfile, "w")) == NULL)
  return;

 /* Start file read and write */
 runtime = 0;
 kfreq = 0;
 npatch = 0;
 while ((fgets (line, MAXLINE, inFilePtr)) != NULL) {

  /* Find total segments used */
  if (strstr (line, "TOTAL SEGMENTS USED") != NULL) {
   sscanf (line, " TOTAL SEGMENTS USED=%d", &nseg);
   ncur = nseg;
   printf ("%.72s\n", line);
   fgets (line, MAXLINE, inFilePtr);
   if (strstr (line, "PATCH") != NULL) {
    sscanf (line, " TOTAL PATCHES USED=%d", &npatch);
    printf ("%.72s\n", line);
   }
   fprintf (fp, "\n \" %s - %3d segments, %4d patches\"\n",
        infile, nseg, npatch);
   fprintf (fp, " Current scaled to %10g Watts\n", normal);
   fprintf (fp, "\n%8s%4s%4s%7s%7s%17s%15s%7s\n", "SEGMENT",
        "TAG", "X", "Y", "Z", "CURRENT", "FREQUENCY",
        "SOURCE");
   fprintf (fp, "%42s%10s%8s\n", "(AMPS)", "(DEG)", "(MHz)");

  /* Skip over segmentation data */
  } else if (strstr (line, "- SEGMENTATION DATA -") != NULL) {
   for (i = 0; i < nseg; i++) fgets (line, MAXLINE, inFilePtr);

  /* Print frequencies to stdout */
  } else if (strstr (line, "- FREQUENCY -") != NULL) {
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   sscanf (line, " FREQUENCY=%f MHZ", &freq);
   printf ("%3s%8.2f%4s\n", "FR", freq, "MHZ");
   kfreq++;

  /* Get antenna input parameters */
  } else if (strstr (line, "- ANTENNA INPUT PARAMETERS -") != NULL) {
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   sscanf (line, "%d %*d %*f %*f %*f %*f %*f %f %*f %*f %f",
        &iextag, &zre, &zim, &power);

  } else if ((strstr (line, " ***** ") != NULL) &&
        (strstr (line, " PT ") != NULL)) {
   sscanf (line, "****** INPUT LINE %*d PT %*d %*d %d %d", &idum, &ncur);
   ncur = ncur - idum + 1;

  } else if (strstr (line, "- CURRENTS AND LOCATION -") != NULL) {
   while (strstr (line, "NO.   NO.") == NULL)
    fgets (line, MAXLINE, inFilePtr);
   if (ncur == 0) ncur = nseg;
   for (i = 0; i < ncur; i++) {
    fgets (line, MAXLINE, inFilePtr);
    sscanf (line, "%d %d %f %f %f %*f %*f %*f %*f %f %f",
         &iseg, &itag, &x, &y, &z, &curM, &curP);
    scale = (float) sqrt ((double) (normal/power));
    curNor = curM * scale;
    fprintf (fp, " %5d%5d%7.2f%7.2f%7.2f %12.4g%7.1f%8.2f%6d\n",
         iseg, itag, x, y, z, curNor, curP, freq, iextag);
   }
  }
```

```
 }
 fclose (fp);
} /* end current */

/*****************************************************************************
 * charge
 *
 * Description: Pulls out selective charge density information from the
 *              NEC output file.
 *
 *****************************************************************************/

static void charge (infile, normal)
    char *infile;
    float normal;
{
 char outfile [132], line [132];
 float freq, power, zre, zim, scale;
 int i, nseg, ncur, iextag, iexcite, incfreq, idum, itag, npatch, nq;
 int kfreq = 0, iwire = 0;
 static char ext [] = ".rq";
 FILE *fp;

 /* Create output filename & open file */
 strcpy(outfile, infile);
 strcat(outfile, ext);
 if ((fp = efopen (outfile, "w")) == NULL)
  return;

 /* Start file read and write */
 npatch = 0;
 while ((fgets (line, MAXLINE, inFilePtr)) != NULL) {

  /* Find total segments used */
  if (strstr (line, "TOTAL SEGMENTS USED") != NULL) {
   sscanf (line, " TOTAL SEGMENTS USED=%d", &nseg);
   ncur = nseg;
   printf ("%.72s\n", line);
   fgets (line, MAXLINE, inFilePtr);
   if (strstr (line, "PATCH") != NULL) {
    sscanf (line, " TOTAL PATCHES USED=%d", &npatch);
    printf ("%.72s\n", line);
   }
   fprintf (fp, "\n \" %s - %3d segments, %4d patches\"\n",
         infile, nseg, npatch);
   fprintf (fp, " Charge scaled to %10g Watts\n", normal);
   fprintf (fp, "\n%8s%4s%4s%7s%7s%12s%13s%7s\n", "SEGMENT",
         "TAG", "X", "Y", "Z", "CHARGE", "FREQUENCY",
         "SOURCE");
   fprintf (fp, "%45s%8s\n", "(COULOMBS)", "(MHz)");

  /* Skip over segmentation data */
  } else if (strstr (line, "- SEGMENTATION DATA -") != NULL) {
   for (i = 0; i < nseg; i++) fgets (line, MAXLINE, inFilePtr);

  /* Print frequencies to stdout */
  } else if (strstr (line, "- FREQUENCY -") != NULL) {
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   sscanf (line, " FREQUENCY=%f MHZ", &freq);
   printf ("%3s%8.2f%4s\n", "FR", freq, "MHZ");
   kfreq++;

  /* Get antenna input parameters */
  } else if (strstr (line, "- ANTENNA INPUT PARAMETERS -") != NULL) {
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   sscanf (line, "%d %*d %*f %*f %*f %*f %*f %f %*f %*f %*f %f",
         &iextag, &zre, &zim, &power);
   scale = (float) sqrt ((double) (normal/power));

  } else if ((strstr (line, " ***** ") != NULL) &&
         (strstr (line, " PT ") != NULL)) {
   sscanf (line, "****** INPUT LINE %*d PT %*d %*d %d %d", &idum, &ncur);
   ncur = ncur - idum + 1;

  /* Skip over currents & location info */
  } else if (strstr (line, "- CURRENTS AND LOCATION -") != NULL) {
   while (strstr (line, "NO.   NO.") == NULL)
    fgets (line, MAXLINE, inFilePtr);

  } else if ((strstr (line, " ***** ") != NULL) &&
         (strstr (line, " PQ ") != NULL)) {
   sscanf (line, "****** INPUT LINE %*d PQ %*d %*d %d %d", &idum, &nq);
   nq = nq - idum + 1;
   if (idum == 0) nq = nseg;

  } else if (strstr (line, "- CHARGE DENSITIES -") != NULL) {
   if (((iextag == iexcite) &&
      (fmod ((double) kfreq, (double) incfreq) == 0)) ||
     (iwire == 0)) {
```

```
      float x, y, z, qNor, qm, qp;
      int iseg;
      char a;

      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      while (sscanf (line, " %d%c %d %f %f %f %*f %*f %*f %f %f",
            &iseg, &a, &itag, &x, &y, &z, &qm, &qp) > 0) {
       qNor = qm * scale;
       if (itag == iwire)
        fprintf (fp, " %5d%c%4d%7.2f%7.2f%7.2f %12.4g%8.2f\n",
            iseg, a, itag, x, y, z, qNor, freq);
       if (iwire == 0)
        fprintf (fp, " %5d%c%4d%7.2f%7.2f%7.2f %12.4g%8.2f%7d\n",
            iseg, a, itag, x, y, z, qNor, freq, iextag);
       fgets (line, MAXLINE, inFilePtr);
      }

     } else {
      for (i = 0; i < nq; i++)
       fgets (line, MAXLINE, inFilePtr);
     }
    }
   }
   fclose (fp);
  } /* end charge */

  /*****************************************************************************
   * pattern
   *
   * Description: Pulls out selective radiation pattern information from
   *          the NEC output file.
   *
   *****************************************************************************/

  static void pattern (infile)
     char *infile;
  {
   char outfile [132], line [132];
   float freq, power, zre, zim;
   int i, nseg, ncur, idum, itag, npatch, ntheta, nphi;
   int kfreq = 0;
   static char ext [] = ".rpt";
   FILE *fp;

   /* Create output filename & open file */
   strcpy(outfile, infile);
   strcat(outfile, ext);
   if ((fp = efopen (outfile, "w")) == NULL)
    return;

   /* Start file read and write */
   npatch = 0;
   while ((fgets (line, MAXLINE, inFilePtr)) != NULL) {

    /* Find total segments used */
    if (strstr (line, "TOTAL SEGMENTS USED") != NULL) {
     sscanf (line, " TOTAL SEGMENTS USED=%d", &nseg);
     ncur = nseg;
     printf ("%.72s\n", line);
     fgets (line, MAXLINE, inFilePtr);
     if (strstr (line, "PATCH") != NULL) {
      sscanf (line, " TOTAL PATCHES USED=%d", &npatch);
      printf ("%.72s\n", line);
     }
     fprintf (fp, "\n \" %s - %3d segments, %4d patches\"\n\n",
          infile, nseg, npatch);

    /* Skip over segmentation data */
    } else if (strstr (line, "- SEGMENTATION DATA -") != NULL) {
     for (i = 0; i < nseg; i++) fgets (line, MAXLINE, inFilePtr);

    } else if ((strstr (line, " ***** ") != NULL) &&
          (strstr (line, " PT ") != NULL)) {
     sscanf (line, "****** INPUT LINE %*d PT %*d %*d %d %d", &idum, &ncur);
     ncur = ncur - idum + 1;

    /* Skip over currents & location info */
    } else if (strstr (line, "- CURRENTS AND LOCATION -") != NULL) {
     if (ncur == 0) ncur = nseg;
     for (i = 0; i < ncur; i++)
      fgets (line, MAXLINE, inFilePtr);

    /* Print frequencies to stdout */
    } else if (strstr (line, "- FREQUENCY -") != NULL) {
     fgets (line, MAXLINE, inFilePtr);
     fgets (line, MAXLINE, inFilePtr);
     sscanf (line, " FREQUENCY=%f MHZ", &freq);
```

```
      fprintf (fp, " FREQUENCY = %9.3f   MHZ\n\n", freq);
      printf ("%3s%8.2f%4s\n", "FR", freq, " MHZ");
      kfreq++;

    /* Get antenna input parameters */
    } else if (strstr (line, "- ANTENNA INPUT PARAMETERS -") != NULL) {
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      sscanf (line, "%d %*f %*f %*f %*f %*f %f %f %*f %*f %f",
            &itag, &zre, &zim, &power);
      fprintf (fp,
            " SOURCE : %4d    %10.3g WATTS    %8.2f%8.2f OHMS\n",
            itag, power, zre, zim);

    } else if (strstr (line, " RP ") != NULL) {
      sscanf (line, " ***** INPUT LINE %*d RP %*d %d %d", &ntheta, &nphi);
      printf ("%72s\n", line);

    } else if (strstr (line, "- RADIATION PATTERNS -") != NULL) {
      float theta, phi, vert, hor, tot, pol, emagTheta, phaTheta,
          emagPhi, phaPhi;
      char apol [15];

      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fprintf (fp, "%9s%6s%10s%7s%17s%17s\n", " Theta ", " Phi ", " Vert ",
            " Hor ", " E-Theta ", " E-Phi ");
      fprintf (fp, "%9s%6s%10s%7s%10s%10s%10s%10s\n",
            " (deg) ", "(deg)", " (dB) ", " (dB) ", "(V/m)", "(deg)",
            "(V/m)", "(deg)");
      while (sscanf (line, " %f %f %f %f %f %f %s %f %f %f %f",
                &theta, &phi, &vert, &hor, &tot, &pol, &apol,
                &emagTheta, &phaTheta, &emagPhi, &phaPhi) > 0) {
        fprintf (fp, " %7.1f%7.1f  %7.2f %7.2f %12.5e%8.2f%12.5e%8.2f\n",
            theta, phi, vert, hor, emagTheta, phaTheta, emagPhi,
            phaPhi);
        fgets (line, MAXLINE, inFilePtr);
      }
      fprintf (fp, "\n");
    }
  }
  fclose (fp);
} /* end pattern */

/*****************************************************************************
 * impedance
 *
 * Description: Pulls out impedance information from the NEC output file
 *              for a single excitation.
 *
 *****************************************************************************/

static void impedance (infile)
    char *infile;
{
  char outfile [132], line [132];
  float freq, power, zre, zim;
  int i, nseg, ncur, idum, itag, npatch;
  int kfreq = 0;
  static char ext [] = ".z";
  FILE *fp;

  /* Create output filename & open file */
  strcpy(outfile, infile);
  strcat(outfile, ext);
  if ((fp = efopen (outfile, "w+")) == NULL)
    return;

  /* Start file read and write */
  npatch = 0;
  while ((fgets (line, MAXLINE, inFilePtr)) != NULL) {

    /* Find total segments used */
    if (strstr (line, "TOTAL SEGMENTS USED") != NULL) {
      sscanf (line, " TOTAL SEGMENTS USED=%d", &nseg);
      ncur = nseg;
      printf ("%.72s\n", line);
      fgets (line, MAXLINE, inFilePtr);
      if (strstr (line, "PATCH") != NULL) {
        sscanf (line, " TOTAL PATCHES USED=%d", &npatch);
        printf ("%.72s\n", line);
      }
      fprintf (fp, "\n \" %s - %3d segments, %4d patches\"\n",
            infile, nseg, npatch);
      fprintf (fp, "\n%7s%6s%8s%8s%16s\n",
            "Freq", "Tag", "R", "X", "GRAPS SMITH");

    /* Skip over segmentation data */
```

```
    } else if (strstr (line, "- SEGMENTATION DATA -") != NULL) {
      for (i = 0; i < nseg; i++) fgets (line, MAXLINE, inFilePtr);

    } else if ((strstr (line, " ***** ") != NULL) &&
           (strstr (line, " PT ") != NULL)) {
      sscanf (line, "***** INPUT LINE %*d PT %*d %*d %d %d", &idum, &ncur);
      ncur = ncur - idum + 1;

    /* Skip over currents & location info */
    } else if (strstr (line, "- CURRENTS AND LOCATION -") != NULL) {
      if (ncur == 0) ncur = nseg;
      for (i = 0; i < ncur; i++)
        fgets (line, MAXLINE, inFilePtr);

    /* Print frequencies to stdout */
    } else if (strstr (line, "- FREQUENCY -") != NULL) {
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      sscanf (line, " FREQUENCY=%f MHZ", &freq);
      printf ("%3s%8.2f%4s\n", "FR", freq, "MHZ");
      kfreq++;

    /* Get antenna input parameters */
    } else if (strstr (line, "- ANTENNA INPUT PARAMETERS -") != NULL) {
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      sscanf (line, "%d %*d %*f %*f %*f %*f %f %f %*f %*f %f",
            &itag, &zre, &zim, &power);
      fprintf (fp, " %6.2f %5d %7.1f %7.1f  %7.2f %7.2f\n",
            freq, itag, zre, zim, zre/50, zim/50);
    }
  }

  /* Extract impedance and admittance from newly created file */
  rewind (fp);
  extract (fp, infile);
  fclose (fp);

} /* end impedance */

/*****************************************************************************
 * extract
 *
 * Description: Program extract reads in a .z impedance file which
 *          has been created by the needsfit program.  It then
 *          extracts impedance and admittance and creates the
 *          .rz and .ra files, respectively.
 *****************************************************************************/

static void extract (zfile, file_name)
    FILE *zfile;
    char *file_name;
{
  FILE *rzfile, *rafile;
  char rz_file[132], ra_file[132];
  static char rz[] = ".rz";
  static char ra[] = ".ra";

  char line1[132], line2[132], line3[132], line4[132];
  int sources[900], max_source, isource, i, ia, npts, j, k;
  float freqs[900], wavelens[900], resis[900], reac[900],
      resis_norm[900], reac_norm[900], conduc[900], suscep[900],
      a, b, c, d, e;

  /* Create new output files */
  strcpy(rz_file,file_name);
  strcpy(ra_file,file_name);
  strcat(rz_file,rz);
  strcat(ra_file,ra);

  if((rzfile = efopen(rz_file,"w")) == NULL)
    exit (1);
  if((rafile = efopen(ra_file,"w")) == NULL)
    exit(1);

  /*  read impedance disk file  */
  fgets(line1, MAXLINE, zfile);
/*  fgets(line2, MAXLINE, zfile); */
  fgets(line3, MAXLINE, zfile);
  fgets(line4, MAXLINE, zfile);
  fgets(line4, MAXLINE, zfile);
  i = 0;
  max_source = 0;
  while (fgets(line4, MAXLINE, zfile) != NULL) {
    sscanf(line4, "%f %d %f %f %f %f", &a, &ia, &b, &c, &d, &e);
    freqs[i] = a;
    wavelens[i] = 300/a;
    sources[i] = ia;
    if (ia > max_source) max_source = ia;
    resis[i] = b;
    reac[i] = c;
```

```
      resis_norm[i] = d;
      reac_norm[i] = e;
      conduc[i] = resis[i]/(resis[i]*resis[i]+reac[i]*reac[i]);
      suscep[i] = -reac[i]/(resis[i]*resis[i]+reac[i]*reac[i]);
      i++;
    }
    npts = i;

    /* write data out */
    fputs(line1, rzfile);
    fputs(line1, rafile);
    fputs(line2, rzfile);
    fputs(line2, rafile);
    fputs(line3, rzfile);
    fputs(line3, rafile);
    for (i = 0; i < max_source; i++) {
      isource = i+1;
      for (k = 0; k < npts; k++) {
        if (sources[k] == isource) {
          fprintf(rzfile,"Source : %4d\n",isource);
          fprintf(rafile,"Source : %4d\n",isource);
          fprintf(rzfile,"                                    Normalized\n");
          fprintf(rzfile," Frequency  Resistance  Reactance  Resistance  Reactance\n");
          fprintf(rzfile,"   (MHz)      (ohms)      (ohms)      (ohms)      (ohms)\n");
          fprintf(rafile," Wavelength  Conductance  Susceptance\n");
          fprintf(rafile,"   (meters)      (mhos)      (mhos)\n");
          for (j = 0; j < npts; j++) {
            if (sources[j] == isource) {
              fprintf(rzfile," %8.2f %11.1f %12.1f %11.2f %12.2f\n",freqs[j],resis[j],reac[j],
                  resis_norm[j], reac_norm[j]);
              fprintf(rafile," %8.2f %13.4f %13.4f\n",wavelens[j],conduc[j],suscep[j]);
            }
          }
          fprintf(rzfile," \n");
          fprintf(rafile," \n");
          break;
        }
      }
    }
    fclose(rzfile);
    fclose(rafile);

} /* end extract */

/*****************************************************************************
 * electric
 *
 * Description: Calls nearField procedure to pull out near electric field
 *              information from NEC file.
 *
 *****************************************************************************/

static void electric (infile, normal)
    char *infile;
    float normal;
{
  nearField (infile, normal, ".me", "- NEAR ELECTRIC FIELDS -", " NE ");

} /* end electric */

/*****************************************************************************
 * magnetic
 *
 * Description: Calls nearField procedure to pull out near magnetic field
 *              information from NEC file.
 *
 *****************************************************************************/

static void magnetic (infile, normal)
    char *infile;
    float normal;
{
  nearField (infile, normal, ".mm", "- NEAR MAGNETIC FIELDS -", " NH ");

} /* end magnetic */

/*****************************************************************************
 * nearField
 *
 * Description: Pulls out near electric/magnetic field information from
 *              the NEC file.  If the last comment cards in the NEC input
 *              deck are as follows:
 *
 *              EXCITATION
 *              [1st antenna excited]
 *              [2nd antenna excited]
 *              [etc]
 *
 *              the antenna names are used in the "***.fld" file created.
 *
 *****************************************************************************/

static void nearField (infile, normal, ext, title, code)
```

```
  char *infile;
  float normal;
  char *ext, *title, *code;
{
 char outfile [132], line [132];
 float freq, power, zre, zim;
 float x0, y0, z0, xstep, ystep, zstep;
 int i, nseg, ncur, idum, itag, npatch;
 int iexcit = 0;
 int ix, iy, iz;
 FILE *fp;
 static char excit [10][72] = {
  "", "", "", "", "", "", "", "", "", ""};

 /* Create output filename & open file */
 strcpy(outfile, infile);
 strcat(outfile, ext);
 if ((fp = efopen (outfile,"w")) == NULL)
  return;

 /* Start file read and write */
 npatch = 0;
 while ((fgets (line, MAXLINE, inFilePtr)) != NULL) {

  /* Find EXCITATION comment */
  if (strstr (line, "EXCITATION") != NULL) {
   while (1) {
    int j;
    char dummy [132], *ptr;

    fgets (line, MAXLINE, inFilePtr);
    if (sscanf (line, " %s\n", dummy) < 1) break;
    ptr = line;
    while (*ptr == (int)' ') ptr++;
    j = strlen (ptr) - 1;
    while (ptr[j] == ' ' || ptr[j] == '\n') j--;
    ptr[++j] = '\0';
    strncpy (excit [i++], ptr, strlen (ptr));
   }

  /* Find total segments used */
  } else if (strstr (line, "TOTAL SEGMENTS USED") != NULL) {
   sscanf (line, " TOTAL SEGMENTS USED=%d", &nseg);
   ncur = nseg;
   printf ("%.72s\n", line);
   fgets (line, MAXLINE, inFilePtr);
   if (strstr (line, "PATCH") != NULL) {
    sscanf (line, " TOTAL PATCHES USED=%d", &npatch);
    printf ("%.72s\n", line);
   }
   fprintf (fp, "\n \" %s - %3d segments, %4d patches\"\n\n",
        infile, nseg, npatch);

  /* Skip over segmentation data */
  } else if (strstr (line, "- SEGMENTATION DATA -") != NULL) {
   for (i = 0; i < nseg; i++) fgets (line, MAXLINE, inFilePtr);

  } else if ((strstr (line, " ***** ") != NULL) &&
        (strstr (line, " PT ") != NULL)) {
   sscanf (line, "***** INPUT LINE %*d PT %*d %*d %d %d", &idum, &ncur);
   ncur = ncur - idum + 1;

  /* Skip over currents & location info */
  } else if (strstr (line, "- CURRENTS AND LOCATION -") != NULL) {
   if (ncur == 0) ncur = nseg;
   for (i = 0; i < ncur; i++)
    fgets (line, MAXLINE, inFilePtr);

  /* Print frequencies to stdout */
  } else if (strstr (line, "- FREQUENCY -") != NULL) {
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   sscanf (line, " FREQUENCY=%f MHZ", &freq);
   fprintf (fp, " FREQUENCY = %9.3f MHZ\n\n", freq);
   printf (" %12s%9.3f%6s\n", "FR", freq, "MHZ");
   iexcit = 0;

  /* Get antenna input parameters */
  } else if (strstr (line, "- ANTENNA INPUT PARAMETERS -") != NULL) {
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   fgets (line, MAXLINE, inFilePtr);
   sscanf (line, "%d %*d %*f %*f %*f %*f %*f %f %*f %*f %*f",
        &itag, &zre, &zim, &power);
   fprintf (fp, " SOURCE : %4d   %10.3g WATTS   %8.2f%8.2f OHMS   %s\n",
         itag, power, zre, zim, excit[iexcit]);
   printf ("%s\n", excit[iexcit++]);

  /* Search for ' NE ' or ' NH ' input line */
  } else if (strstr (line, code) != NULL) {
   char inputLine [132];
   char template [] =
```

```
        " ***** INPUT LINE %%*d%s%%*d %%d %%d %%d %%f %%f %%f %%f %%f";

      sprintf (inputLine, template, code);
      sscanf (line, inputLine,
           &ix, &iy, &iz, &x0, &y0, &z0, &xstep, &ystep, &zstep);

    /* Search for ' NEAR ELECTRIC FIELD ' or ' NEAR MAGNETIC FIELD ' */
    } else if (strstr (line, title) != NULL) {
      float scale, ii;
      char card [132];
      char template [] =
        " %s%%5d%%5d%%5d    %%8.2f%%8.2f%%8.2f    %%8.2f%%8.2f%%8.2f\n";

      sprintf (card, template, code);
      printf (card, ix, iy, iz, x0, y0, z0, xstep, ystep, zstep);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fgets (line, MAXLINE, inFilePtr);
      fprintf (fp,
           " In the following: average field normalized to %10g WATTS\n",
           normal);
      if (strcmp(ext,".mm") == 0)
        fprintf (fp, "%7s %7s %7s  %7s %7s %7s  %8s\n", " X ", " Y ",
             " Z ", " Hx ", " Hy ", " Hz ", " H_total");
      else
        fprintf (fp, "%7s %7s %7s  %7s %7s %7s  %8s\n", " X ", " Y ",
             " Z ", " Ex ", " Ey ", " Ez ", " E_total");
      scale = (float) sqrt ((double) normal/power);
      ii = ix * iy * iz;
      for (i = 0; i < ii; i++) {
        float x, y, z, ex, ey, ez, e_av,
             ex_av, ex_nor, ey_av, ey_nor,
             ez_av, ez_nor, e_total;

        fgets (line, MAXLINE, inFilePtr);
        sscanf (line, "%f %f %f %f %*f %f %*f %f",
             &x, &y, &z, &ex, &ey, &ez);

        ex_av = ex / sqrt (2.0);
        ex_nor = ex_av * scale;
        ey_av = ey / sqrt (2.0);
        ey_nor = ey_av * scale;
        ez_av = ez / sqrt (2.0);
        ez_nor = ez_av * scale;
        e_av = sqrt ((double) (ex*ex + ey*ey + ez*ez) / 2);
        e_total = e_av * scale;

        fprintf (fp, "%7.2f %7.2f %7.2f  %7.2f %7.2f %7.2f  %8.2f\n",
             x, y, z, ex_nor, ey_nor, ez_nor, e_total);
      }
      fprintf (fp, "\n");
    }
  }
  fclose (fp);
} /* end nearField */
```

A.18 necdisp.c:

necdisp.c:

```
#include <Xm/DrawingA.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/Frame.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/Scale.h>
#include <Xm/Separator.h>
#include <Xm/List.h>
#include <Xm/SelectioB.h>
#include <Xm/PanedW.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "xgraphics.h"
#include "cFileMenu.h"
#include "actionArea.h"

#define MAXLINE 120

#define degrees_to_radians 0.017453292
#define cdr 0.017453292

extern int NumWires, NumSegs;
extern float Xmin, Xmax, Ymin, Ymax, Zmin, Zmax, Xshift, Yshift, Zshift, Scale,
                SegMin, SegMax, RadMin, RadMax, SrMin, SrMax;
extern float *Xnode1, *Ynode1, *Znode1, *Xnode2, *Ynode2, *Znode2;
extern float *Segs, *Rads, *SrRatio;

/* Indexes into global arrays */
extern int *Jseg, *Jwire, *Iseg, *Irad, *Isr, *Icon, *Iseglg, *Iradlg, *Isrlg;
extern int *Idiag;

static float ***tdbdata;
static float ***pdbdata;
static float ***ethdata;
static float ***ephdata;
static float **ethphase;
static float **ephphase;
/*
static float tdbdata[360][90][3];
static float pdbdata[360][90][3];
static float ethdata[360][90][3];
static float ephdata[360][90][3];
static float ethphase[360][90];
static float ephphase[360][90];
*/
static float phase;

static float minval[16], maxval[16];
static int **ival;
static float **ne_val;
static float **nh_val;
static XgPoint *ne_points;
static XgPoint *nh_points;
static XgPoint *bgn_points;
static XgPoint *end_points;
/*
static int ival[15][3000];
static float ne_val[2][3000];
static XgPoint ne_points[3000];
static XgPoint bgn_points[3000],end_points[3000];
*/
static int maxthetas, maxphis;
static XgPoint axes[4];
static int numsegs;
static int numnepts;
static int numnhpts;

static char in_file_ne[132];
static char in_file_nh[132];
static char in_file_pt[132];
static char in_file_cr[132];
static char in_file_q[132];

static XgDevice xgd1;

static void initialize();
static void destroyCB();
static void DrawCallback();
static void PushCallback();
static void PushCallback2();
static void ScaleCallback();

static char *buttonLabels [] = {"Plan", "Elevation", "Bow"};
static char *labels[] = {"Toggle Axes", "Reset", "Print", "Quit"};
```

```
static char *zlabels[] = {"X","Y","Scale"};

/* Drawing data */
typedef struct _drawData {
  int     type;             /* Drawing type */
  XgDevice xgd;             /* Xgraphics device for drawing */
  Widget  az_scale, el_scale,  /* Scale widgets */
                x_scale, y_scale,
                zoom_scale;
  int     source;
  float   freq;
} DrawData;

static int wireIndex;
static void updateCB();
Boolean DRAW = False;
static Widget popupShell = NULL;
extern Widget straightWiresShell;
extern Widget nodeCoordShell;

/*****************************************************************************
 *  necDisplay
 *****************************************************************************/

void necDisplay (full_name, plttype, source, selfreq)
    char  *full_name;
    int   plttype;
    int   source;
    float selfreq;
{
 Display    *display;
 Widget     shell, Main, frame, form, rowCol, button,
                drawing1, separator, scale, rowCol2,
                az_scale, el_scale, x_scale, y_scale, zoom_scale;
 int        i,value;
 XgOrient   orient;
 XgZoom     zoom;
 XgExtents  extents;
 DrawData   *drawData;
 char file_name[132];
 char *toptitle;
 char title[132];
 char *ptr;
 static char me[] = ".me";
             static char mm[] = ".mm";
 static char rpt[] = ".rpt";
 static char rcr[] = ".rcr";
 static char rq[] = ".rq";
 static char *pretitles [] = { "Diagnostics: ",
  "Segmentation: ", "Wire Radius: ", "Segment to Radius Ratio: ",
  "Wire Connections: ", "Segmentation (log scale): ",
  "Wire Radius (log scale): ", "Segment to Radius Ratio (log scale): ",
  "Current Magnitude: ", "Current Magnitude (log scale): ",
  "Current Phase: ", "Charge: ", "Charge (log scale): ",
  "Ez normalized: ", "E normalized: ",
                              "Hx normalized: ", "Hy normalized: ", "E-theta: ",
  "E-theta (dB): ", "E-phi: ", "E-phi (dB): "};
 Arg args[2];
 extern Widget topLevel;

 if (plttype > 7) {
 /* get in filename */
  strcpy(file_name,full_name);
  ptr = strrchr(file_name,'.');
  if (ptr) *(ptr) = '\0';
 }

 /* check to see whether we are doing currents */
 if ((plttype == 8)||(plttype == 9)||(plttype == 10)) {
   strcpy(in_file_cr,file_name);
   strcat(in_file_cr,rcr);
 }
 /* check to see whether we are doing charges */
 if ((plttype == 11)||(plttype == 12)) {
   strcpy(in_file_q,file_name);
   strcat(in_file_q,rq);
 }
 /* check to see whether we are doing near electric fields */
 if ((plttype == 13)||(plttype == 14)) {
   strcpy(in_file_ne,file_name);
   strcat(in_file_ne,me);
 }
 /* check to see whether we are doing near magnetic fields */
 if ((plttype == 15)||(plttype == 16)) {
   strcpy(in_file_nh,file_name);
   strcat(in_file_nh,mm);
 }
 /* check to see whether we are doing far field patterns */
 if ((plttype == 17)||(plttype == 18)||(plttype == 19)||(plttype == 20)) {
   strcpy(in_file_pt,file_name);
   strcat(in_file_pt,rpt);
 }
```

```
/* Create the window title */
i = strlen (pretitles[plttype]);
strncpy (title, pretitles[plttype], i);
title[i] = '\0';
strcat(title, full_name);
toptitle = title;
initialize(source, selfreq, plttype);

XtSetArg (args[0], XmNtitle, title);
shell = XtCreatePopupShell ("XGraphics", topLevelShellWidgetClass,
                                                topLevel, args, 1);
XtVaSetValues (shell, XmNdeleteResponse, XmDESTROY, NULL);
XtAddCallback (shell, XmNdestroyCallback, destroyCB, NULL);

display = XtDisplay(shell);

Main = XtVaCreateManagedWidget
 ("Main",xmFormWidgetClass,shell,
  XmNx, 15,
  XmNy, 15,
  NULL);

frame = XtVaCreateManagedWidget
 ("frame", xmFrameWidgetClass, Main,
  XmNtopAttachment, XmATTACH_FORM,
  XmNleftAttachment, XmATTACH_FORM,
  XmNleftOffset,    15,
  XmNtopOffset,     15,
  XmNshadowType,      XmSHADOW_OUT,
  NULL);

form = XtVaCreateManagedWidget
 ("form", xmFormWidgetClass, frame, NULL);

rowCol = XtVaCreateManagedWidget
 ("rowCol", xmRowColumnWidgetClass, form,
  XmNtopAttachment, XmATTACH_FORM,
  XmNleftAttachment, XmATTACH_FORM,
  XmNmarginHeight,  1,
  XmNmarginWidth,   1,
  XmNspacing,       1,
  NULL);

for (i = 0; i < XtNumber (buttonLabels); i++) {
 button = XtVaCreateManagedWidget
  ("button", xmPushButtonWidgetClass, rowCol,
   XtVaTypedArg, XmNlabelString,
   XmRString, buttonLabels[i], strlen (buttonLabels[i]) + 1,
   NULL);
 XtAddCallback(button, XmNactivateCallback, PushCallback2, (XtPointer)i);
}

az_scale = XtVaCreateManagedWidget
 ("az_scale", xmScaleWidgetClass, form,
  XmNtopAttachment,  XmATTACH_WIDGET,
  XmNtopWidget,      rowCol,
  XmNtopOffset,     25,
  XmNleftAttachment, XmATTACH_FORM,
  XmNwidth,         150,
  XmNminimum,       -180,
  XmNmaximum,       180,
  XmNvalue,         0,
  XmNshowValue,     True,
  XmNorientation,   XmHORIZONTAL,
  XtVaTypedArg, XmNtitleString, XmRString, "Azimuth", 8,
  NULL);
XtAddCallback(az_scale,XmNvalueChangedCallback,ScaleCallback,(XtPointer)0);
XtAddCallback(az_scale,XmNdragCallback,ScaleCallback,(XtPointer)0);

el_scale = XtVaCreateManagedWidget
 ("el_scale", xmScaleWidgetClass, form,
  XmNtopAttachment,  XmATTACH_WIDGET,
  XmNtopWidget,      az_scale,
  XmNtopOffset,     15,
  XmNleftAttachment, XmATTACH_FORM,
  XmNwidth,         150,
  XmNminimum,       -90,
  XmNmaximum,       90,
  XmNvalue,         0,
  XmNshowValue,     True,
  XmNorientation,   XmHORIZONTAL,
  XtVaTypedArg, XmNtitleString, XmRString,"Elevation", 10,
  NULL);

XtAddCallback(el_scale,XmNvalueChangedCallback,ScaleCallback,(XtPointer)1);
XtAddCallback(el_scale,XmNdragCallback,ScaleCallback,(XtPointer)1);

value = 50;
scale = el_scale;
for(i=0;i<XtNumber(zlabels);i++) {
 if(i == 2) value = 100;
 scale = XtVaCreateManagedWidget
  ("scale", xmScaleWidgetClass, form,
```

```
      XmNtopAttachment,   XmATTACH_WIDGET,
      XmNtopWidget,       scale,
      XmNtopOffset,       15,
      XmNleftAttachment,  XmATTACH_FORM,
      XmNwidth,           150,
      XmNminimum,         0,
      XmNmaximum,         100,
      XmNvalue,           value,
      XmNshowValue,       True,
      XmNorientation,     XmHORIZONTAL,
      XtVaTypedArg, XmNtitleString, XmRString, zlabels[i], 6,
      NULL);
    XtAddCallback (scale, XmNvalueChangedCallback, ScaleCallback,
                   (XtPointer)(i+2));
    XtAddCallback (scale, XmNdragCallback, ScaleCallback,
                   (XtPointer)(i+2));
    if(i==0) x_scale = scale;
    if(i==1) y_scale = scale;
    if(i==2) zoom_scale = scale;
  }

  frame = XtVaCreateManagedWidget
    ("frame", xmFrameWidgetClass, Main,
     XmNtopAttachment,   XmATTACH_FORM,
     XmNtopOffset,       15,
     XmNrightAttachment, XmATTACH_FORM,
     XmNrightOffset,     15,
     XmNleftAttachment,  XmATTACH_WIDGET,
     XmNleftWidget,      frame,
     XmNleftOffset,      15,
     XmNshadowType,      XmSHADOW_OUT,
     XmNmarginWidth,     10,
     XmNmarginHeight,    10,
     NULL);

  drawing1 = XtVaCreateManagedWidget
    ("button", xmDrawingAreaWidgetClass, frame,
     XmNwidth,           500,
     XmNheight,          500,
     XtVaTypedArg,XmNbackground,XmRString,"black",4,
     XtVaTypedArg,XmNforeground,XmRString,"white",4,
     NULL);

  separator = XtVaCreateManagedWidget
    ("sep", xmSeparatorWidgetClass, Main,
     XmNtopAttachment,   XmATTACH_WIDGET,
     XmNtopWidget,       frame,
     XmNtopOffset,       10,
     XmNrightAttachment, XmATTACH_FORM,
     XmNleftAttachment,  XmATTACH_FORM,
     NULL);

  rowCol2 = XtVaCreateManagedWidget
    ("rowCol", xmRowColumnWidgetClass, Main,
     XmNtopAttachment,  XmATTACH_WIDGET,
     XmNtopWidget,      separator,
     XmNtopOffset,      10,
     XmNleftAttachment, XmATTACH_FORM,
     XmNleftOffset,     10,
     XmNrightAttachment,XmATTACH_FORM,
     XmNbottomAttachment,XmATTACH_FORM,
     XmNbottomOffset,   10,
     XmNmarginHeight,   1,
     XmNmarginWidth,    1,
     XmNspacing,        1,
     XmNorientation,    XmHORIZONTAL,
     NULL);

  for(i=0;i<XtNumber(labels);i++) {
    button = XtVaCreateManagedWidget
      ("button", xmPushButtonWidgetClass, rowCol2,
       XtVaTypedArg,XmNlabelString,XmRString,labels[i],20,
       NULL);
    XtAddCallback(button,XmNactivateCallback,PushCallback,(XtPointer)i);
  }

  XtPopup (shell, XtGrabNone);

  XgOpenXGraphics();
  xgd1 = XgOpenWidget(drawing1);
  XtAddCallback(drawing1, XmNinputCallback, DrawCallback, (XtPointer) NULL);

  /* Create structure for storing drawing data */
  drawData = (DrawData *) XtMalloc (sizeof (DrawData));
  drawData->type = plttype;
  drawData->xgd = xgd1;
  drawData->az_scale = az_scale;
  drawData->el_scale = el_scale;
  drawData->x_scale = x_scale;
  drawData->y_scale = y_scale;
  drawData->zoom_scale = zoom_scale;
  drawData->source = source;
  drawData->freq = selfreq;
```

```
  if (plttype < 17)
    XtAddEventHandler(shell, FocusChangeMask, False,
      (XtEventHandler)updateCB, (XtPointer)drawData);

  XtVaSetValues (shell, XmNuserData, drawData, NULL);
  XtVaSetValues (form, XmNuserData, drawData, NULL);
  XtVaSetValues (rowCol2, XmNuserData, drawData, NULL);
  XtVaSetValues (drawing1, XmNuserData, drawData, NULL);

  XgInquireOrthographicView(&orient,&zoom,&extents);
  orient.az = 0;
  orient.el = 0;
  zoom.xcenter = 0.5;
  zoom.ycenter = 0.5;
  zoom.scale = 1.0;
  extents.ymin = -2.0;
  extents.ymax = 2.0;
  XgSetOrthographicView(&orient,&zoom,&extents);
  XgSetCurrentDevice(xgd1);

  /* Draw graphics */
  if (plttype < 17)
    DrawGeo (plttype);
  else
    DrawPat (plttype);
  if ((plttype == 13)||(plttype == 14))
    DrawNE (plttype);
  if ((plttype == 15)||(plttype == 16))
    DrawNH (plttype);

}
/*****************************************************************************/

void initialize (source, selfreq, plttype)
    int source;
    float selfreq;
    int plttype;
{
  FILE *infile_ne;
  FILE *infile_nh;
  FILE *infile_pt;
  FILE *infile_cr;
  FILE *infile_q;
  char line[120];
  char lookfor1[23];
  char lookfor2[15];
  int numwires;
  int i,j,ia,ib,ic;
  char ch;
  float a,b,c,d,e,f,o,p;
  float xmin,xmax,ymin,ymax,zmin,zmax;
  float xcen,ycen,zcen;
  float mscale;
  float Ez_max,E_max;
  float Hx_max,Hy_max;
  int ipts, iphis, ithetas;
  float tdbmax, pdbmax, ethmax, ephmax;
  float theta, phi, tdb, pdb, ethmag, ethph, ephmag, ephph;
  float oldtheta, oldphi;
  float *ccdata;
/*
  float ccdata[3000];
*/
  if (plttype < 17) {
    /* handle currents, if necessary */
    if ((plttype == 8)||(plttype == 9)||(plttype == 10)) {
      if ((infile_cr = fopen(in_file_cr,"r")) == (FILE*)0) {
              printf("\n\t\t File %s could not be opened. \n",in_file_cr);
              exit(1);
      }
                              ccdata = (float *)XtCalloc(NumSegs,sizeof(float));
      /* dump first 6 lines */
      fgets(line, MAXLINE, infile_cr);
      fgets(line, MAXLINE, infile_cr);
      fgets(line, MAXLINE, infile_cr);
      fgets(line, MAXLINE, infile_cr);
      fgets(line, MAXLINE, infile_cr);
      fgets(line, MAXLINE, infile_cr);
      /* now read in current data */
      i = 0;
      while ((fscanf(infile_cr,"%d %d %f %f %f %f %f %f %d",
                              &ia,&ib,&a,&b,&c,&d,&e,&f,&ic)) != EOF) {
              if ((source == ic)&&(selfreq == f)) {
                switch (plttype)
                 {
                 case 8: ccdata[i] = d;
                   if (i == 0) { minval[7] = d; maxval[7] = d; }
                   else {
                              if (d < minval[7]) minval[7] = d;
                              if (d > maxval[7]) maxval[7] = d;
                   }
                   break;
                 case 9: ccdata[i] = d;
```

```
                if (i == 0) { minval[8] = d; maxval[8] = d; }
                else {
                        if (d < minval[8]) minval[8] = d;
                        if (d > maxval[8]) maxval[8] = d;
                }
                break;
               case 10: while (e < 0) e = e + 360.0;
                while (e >= 360.0) e = e - 360.0;
                ccdata[i] = e/60.0;
               }
              i++;
             }
  }
  fclose(infile_cr);
 }
 /* handle charges, if necessary */
 if ((plttype == 11)||(plttype == 12)) {
  if ((infile_q = fopen(in_file_q,"r")) == (FILE*)0) {
            printf("\n\t\t File %s could not be opened. \n",in_file_q);
            exit(1);
  }
  /* dump first 6 lines */
  ccdata = (float *)XtCalloc(NumSegs,sizeof(float));
  fgets(line, MAXLINE, infile_q);
  fgets(line, MAXLINE, infile_q);
  fgets(line, MAXLINE, infile_q);
  fgets(line, MAXLINE, infile_q);
  fgets(line, MAXLINE, infile_q);
  fgets(line, MAXLINE, infile_q);
  /* now read in charge data */
  i = 0;
  while ((fscanf(infile_q,"%5d%c %d %f %f %f %f %f %d",
                                &ia,&ch,&ib,&a,&b,&c,&d,&e,&ic)) != EOF) {
           if ((source == ic)&&(selfreq == e)) {
            if (ch == 'E') i = i-1;
            ccdata[i] = d;
            switch (plttype)
             {
             case 11: if (i == 0) { minval[10] = d; maxval[10] = d; }
             else {
              if (d < minval[10]) minval[10] = d;
              if (d > maxval[10]) maxval[10] = d;
             }
              break;
             case 12: if (i == 0) { minval[11] = d; maxval[11] = d; }
             else {
              if (d < minval[11]) minval[11] = d;
              if (d > maxval[11]) maxval[11] = d;
             }
              break;
             }
            i++;
           }
  }
  fclose(infile_q);
}

/* now go read geometry data */
numwires = NumWires;
numsegs = NumSegs;
/* now go allocate the memory */
ival = (int **)XtCalloc(17,sizeof(int *));
for (i=0; i<17; ++i)
{
 ival[i] = (int *)XtCalloc(NumSegs,sizeof(int));
}
bgn_points = (XgPoint *)XtCalloc(NumSegs,sizeof(XgPoint));
end_points = (XgPoint *)XtCalloc(NumSegs,sizeof(XgPoint));

xmin = Xmin; xmax = Xmax;
ymin = Ymin; ymax = Ymax;
zmin = Zmin; zmax = Zmax;
xcen = (xmin+xmax)/2.0;
ycen = (ymin+ymax)/2.0;
zcen = (zmin+zmax)/2.0;
mscale = Scale;
mscale = mscale * 1.0;
minval[0] = SegMin;
minval[4] = SegMin;
maxval[0] = SegMax;
maxval[4] = SegMax;
minval[1] = RadMin;
minval[5] = RadMin;
maxval[1] = RadMax;
maxval[5] = RadMax;
minval[2] = SrMin;
minval[6] = SrMin;
maxval[2] = SrMax;
maxval[6] = SrMax;

for (i=0; i<numsegs; i++) {
 switch (plttype) {
 case 0:  ival[0][i] = ldiag[i];
```

```
            break;
  case 1:  ival[1][i] = lseg[i];
            break;
  case 2:  ival[2][i] = lrad[i];
            break;
  case 3:  ival[3][i] = lsr[i];
            break;
  case 4:  ival[4][i] = lcon[i];
            break;
  case 5:  ival[5][i] = lseglg[i];
            break;
  case 6:  ival[6][i] = lradlg[i];
            break;
  case 7:  ival[7][i] = lsrlg[i];
            break;
  case 8:
            if (minval[7] == maxval[7]) ival[8][i] = 0;
            else {
              ival[8][i] = (ccdata[i]-minval[7])/(maxval[7]-minval[7])*7;
              if (ccdata[i]==maxval[7]) ival[8][i] = 6;
            }
            break;
  case 9:
            if (minval[8] == maxval[8]) ival[9][i] = 0;
            else {
              ival[9][i] = (log10(minval[8]/ccdata[i])/log10(minval[8]/maxval[8]))*7;
              if (ccdata[i]==maxval[8]) ival[9][i] = 6;
            }
            break;
  case 10:
            ival[10][i] = ccdata[i];
            break;
  case 11:
            if (minval[10] == maxval[10]) ival[11][i] = 0;
            else {
              ival[11][i] = (ccdata[i]-minval[10])/(maxval[10]-minval[10])*7;
              if (ccdata[i]==maxval[10]) ival[11][i] = 6;
            }
            break;
  case 12:
            if (minval[11] == maxval[11]) ival[12][i] = 0;
            else {
              ival[12][i] = (log10(minval[11]/ccdata[i])/log10(minval[11]/maxval[11]))*7;
              if (ccdata[i]==maxval[11]) ival[12][i] = 6;
            }
            break;
  case 13:
            ival[13][i] = 7;
            break;
  case 14:
            ival[14][i] = 7;
            break;
  case 15:
            ival[15][i] = 7;
            break;
  case 16:
            ival[16][i] = 7;
            break;

 }
 bgn_points[i].x = -(Ynode1[i]-ycen)*mscale;
 bgn_points[i].y = -(Xnode1[i]-xcen)*mscale;
 bgn_points[i].z = Znode1[i]*mscale;
 end_points[i].x = -(Ynode2[i]-ycen)*mscale;
 end_points[i].y = -(Xnode2[i]-xcen)*mscale;
 end_points[i].z = Znode2[i]*mscale;

}
if ((plttype > 7)&&(plttype < 13)) XtFree((char *) ccdata);
axes[0].x = 0.0;
axes[0].y = xcen*mscale;
axes[0].z = 0.0;
axes[1].x = 0.0;
axes[1].y = xcen*mscale-4.0;
axes[1].z = 0.0;
axes[2].x = -4.0;
axes[2].y = xcen*mscale;
axes[2].z = 0.0;
axes[3].x = 0.0;
axes[3].y = xcen*mscale;
axes[3].z = 4.0;
}
/* put in near electric fields if necessary */
if ((plttype == 13)||(plttype == 14)) {
 if ((infile_ne = fopen(in_file_ne,"r")) == (FILE*)0) {
  printf("\n\t\t File %s could not be opened. \n",in_file_ne);
  exit(1);
 }
 /* first look for frequency */
 sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
 fgets(line, MAXLINE, infile_ne);
 *(line + 22) = '\0';
 while(strcmp(lookfor1,line) != 0)
```

```
  {
            if (fgets(line, MAXLINE, infile_ne) == NULL) {
              printf("\n REACHED END of FILE!");
              exit(1);
            }
            *(line + 22) = '\0';
  }
 /* now look for source */
 sprintf(lookfor2," SOURCE :%5d",source);
 fgets(line, MAXLINE, infile_ne);
 *(line + 14) = '\0';
 while(strcmp(lookfor2,line) != 0)
  {
            if (fgets(line, MAXLINE, infile_ne) == NULL) {
              printf("\n REACHED END of FILE!");
              exit(1);
            }
            *(line + 14) = '\0';
  }
 /* now read in data */
 fgets(line, MAXLINE, infile_ne);
 fgets(line, MAXLINE, infile_ne);
 i = 0;
 Ez_max = 0.0;
 E_max = 0.0;
 while ((fscanf(infile_ne,"%f %f %f %f %f %f %f",
                          &a,&b,&c,&d,&e,&o,&p)) == 7) {
   if (o > Ez_max) Ez_max = o;
   if (p > E_max) E_max = p;
   i++;
 }
 numnepts = i;
 minval[plttype-1] = 0;
 if (plttype == 13) maxval[12] = Ez_max;
 if (plttype == 14) maxval[13] = E_max;
 /* set aside memory */
 ne_val = (float **)XtCalloc(2,sizeof(float *));
 for (i=0; i<2; ++i)
 {
 ne_val[i] = (float *)XtCalloc(numnepts,sizeof(float));
 }
 ne_points = (XgPoint *)XtCalloc(numnepts,sizeof(XgPoint));

 rewind(infile_ne);
 /* first look for frequency */
 sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
 fgets(line, MAXLINE, infile_ne);
 *(line + 22) = '\0';
 while(strcmp(lookfor1,line) != 0)
  {
            if (fgets(line, MAXLINE, infile_ne) == NULL) {
              printf("\n REACHED END of FILE!");
              exit(1);
            }
            *(line + 22) = '\0';
  }
 /* now look for source */
 sprintf(lookfor2," SOURCE :%5d",source);
 fgets(line, MAXLINE, infile_ne);
 *(line + 14) = '\0';
 while(strcmp(lookfor2,line) != 0)
  {
            if (fgets(line, MAXLINE, infile_ne) == NULL) {
              printf("\n REACHED END of FILE!");
              exit(1);
            }
            *(line + 14) = '\0';
  }

 /* now read in data */
 fgets(line, MAXLINE, infile_ne);
 fgets(line, MAXLINE, infile_ne);
 for (i=0; i<numnepts; i++) {
  fscanf(infile_ne,"%f %f %f %f %f %f %f",
                &a,&b,&c,&d,&e,&o,&p);
  if (plttype == 13) ne_val[0][i] = o/Ez_max * 7.0;
  if (plttype == 14) ne_val[1][i] = p/E_max * 7.0;
  ne_points[i].x = (-b+ycen)*mscale;
  ne_points[i].y = (-a+xcen)*mscale;
  ne_points[i].z = c*mscale;
  /*
            SPH_setMarkerSizeScaleFactor (val/7.0);
            if (val < 1.0) SPH_setMarkerColor( blue );
            else if (val < 2.0) SPH_setMarkerColor( cyan );
            else if (val < 3.0) SPH_setMarkerColor( green );
            else if (val < 4.0) SPH_setMarkerColor( yellow );
            else if (val < 5.0) SPH_setMarkerColor( orange );
            else if (val < 6.0) SPH_setMarkerColor( red );
            else SPH_setMarkerColor ( white );
            tempv[X] = (a-xcen)*mscale;
            tempv[Y] = (c-zcen)*mscale;
            tempv[Z] = (-b+ycen)*mscale;
            SPH_polyMarker( 1, &tempv);
```

```
            */
 }
 fclose(infile_ne);
}
/* put in near magnetic fields if necessary */
if ((plttype == 15)||(plttype == 16)) {
 if ((infile_nh = fopen(in_file_nh,"r")) == (FILE*)0) {
  printf("\n\t\t File %s could not be opened. \n",in_file_nh);
  exit(1);
 }
 /* first look for frequency */
 sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
 fgets(line, MAXLINE, infile_nh);
 *(line + 22) = '\0';
 while(strcmp(lookfor1,line) != 0)
  {
            if (fgets(line, MAXLINE, infile_nh) == NULL) {
             printf("\n REACHED END of FILE!");
             exit(1);
            }
            *(line + 22) = '\0';
  }
 /* now look for source */
 sprintf(lookfor2," SOURCE :%5d",source);
 fgets(line, MAXLINE, infile_nh);
 *(line + 14) = '\0';
 while(strcmp(lookfor2,line) != 0)
  {
            if (fgets(line, MAXLINE, infile_nh) == NULL) {
             printf("\n REACHED END of FILE!");
             exit(1);
            }
            *(line + 14) = '\0';
  }
 /* now read in data */
 fgets(line, MAXLINE, infile_nh);
 fgets(line, MAXLINE, infile_nh);
 i = 0;
 Hx_max = 0.0;
 Hy_max = 0.0;
 while ((fscanf(infile_nh,"%f %f %f %f %f %f %f",
                          &a,&b,&c,&d,&e,&o,&p)) == 7) {
  if (d > Hx_max) Hx_max = d;
  if (e > Hy_max) Hy_max = e;
  i++;
 }
 numnhpts = i;
 minval[plttype-1] = 0;
 if (plttype == 15) maxval[14] = Hx_max;
 if (plttype == 16) maxval[15] = Hy_max;
 /* set aside memory */
 nh_val = (float **)XtCalloc(2,sizeof(float *));
 for (i=0; i<2; ++i)
 {
 nh_val[i] = (float *)XtCalloc(numnhpts,sizeof(float));
 }
 nh_points = (XgPoint *)XtCalloc(numnhpts,sizeof(XgPoint));

 rewind(infile_nh);
 /* first look for frequency */
 sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
 fgets(line, MAXLINE, infile_nh);
 *(line + 22) = '\0';
 while(strcmp(lookfor1,line) != 0)
  {
            if (fgets(line, MAXLINE, infile_nh) == NULL) {
             printf("\n REACHED END of FILE!");
             exit(1);
            }
            *(line + 22) = '\0';
  }
 /* now look for source */
 sprintf(lookfor2," SOURCE :%5d",source);
 fgets(line, MAXLINE, infile_nh);
 *(line + 14) = '\0';
 while(strcmp(lookfor2,line) != 0)
  {
            if (fgets(line, MAXLINE, infile_nh) == NULL) {
             printf("\n REACHED END of FILE!");
             exit(1);
            }
            *(line + 14) = '\0';
  }

 /* now read in data */
 fgets(line, MAXLINE, infile_nh);
 fgets(line, MAXLINE, infile_nh);
 for (i=0; i<numnhpts; i++) {
  fscanf(infile_nh,"%f %f %f %f %f %f %f",
              &a,&b,&c,&d,&e,&o,&p);
  if (plttype == 15) nh_val[0][i] = d/Hx_max * 7.0;
  if (plttype == 16) nh_val[1][i] = e/Hy_max * 7.0;
  nh_points[i].x = (-b+ycen)*mscale;
```

```
    nh_points[i].y = (-a+xcen)*mscale;
    nh_points[i].z = c*mscale;
    /*
                SPH_setMarkerSizeScaleFactor (val/7.0);
                if (val < 1.0) SPH_setMarkerColor( blue );
                else if (val < 2.0) SPH_setMarkerColor( cyan );
                else if (val < 3.0) SPH_setMarkerColor( green );
                else if (val < 4.0) SPH_setMarkerColor( yellow );
                else if (val < 5.0) SPH_setMarkerColor( orange );
                else if (val < 6.0) SPH_setMarkerColor( red );
                else SPH_setMarkerColor ( white );
                tempv[X] = (a-xcen)*mscale;
                tempv[Y] = (c-zcen)*mscale;
                tempv[Z] = (-b+ycen)*mscale;
                SPH_polyMarker( 1, &tempv);
                */
  }
  fclose(infile_nh);
}
/* now go read pattern data, if necessary */
if (plttype >= 17) {
  if ((infile_pt = fopen(in_file_pt,"r")) == (FILE*)0) {
    printf("\n\tt File %s could not be opened. \n",in_file_pt);
    exit(1);
  }
  maxthetas = 0;
  maxphis = 0;
  tdbmax = -1000.0;
  pdbmax = -1000.0;
  ethmax = 0.0;
  ephmax = 0.0;
  /* set up maximum ranges for theta and phi */
  /* first look for frequency */
  sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
  fgets(line, MAXLINE, infile_pt);
  *(line + 22) = '\0';
  while(strcmp(lookfor1,line) != 0)
   {
              if (fgets(line, MAXLINE, infile_pt) == NULL) {
                printf("\n REACHED END of FILE!");
                exit(1);
              }
              *(line + 22) = '\0';
   }
  /* now look for source */
  sprintf(lookfor2," SOURCE :%5d",source);
  fgets(line, MAXLINE, infile_pt);
  *(line + 14) = '\0';
  while(strcmp(lookfor2,line) != 0)
   {
              if (fgets(line, MAXLINE, infile_pt) == NULL) {
                printf("\n REACHED END of FILE!");
                exit(1);
              }
              *(line + 14) = '\0';
   }

  /* dump next two lines */
  fgets(line,MAXLINE,infile_pt);
  fgets(line,MAXLINE,infile_pt);
  iphis = 1;
  ithetas = 1;
  i=1;
  fscanf(infile_pt, "%f %f %f %f %f %f %f %f", &theta, &phi, &tdb,
                &pdb, &ethmag, &ethph, &ephmag, &ephph);
  oldtheta = theta;
  oldphi = phi;
  if (tdb > tdbmax) tdbmax = tdb;
  if (pdb > pdbmax) pdbmax = pdb;
  if (ethmag > ethmax) ethmax = ethmag;
  if (ephmag > ephmax) ephmax = ephmag;
  while ((fscanf(infile_pt,"%f %f %f %f %f %f %f %f",&theta, &phi, &tdb,
                                &pdb, &ethmag, &ethph, &ephmag, &ephph)) == 8) {
   if (theta > oldtheta)
              {
                ithetas++;
                oldtheta = theta;
              }
   if (phi > oldphi)
              {
                iphis++;
                oldphi = phi;
              }
   if (tdb > tdbmax) tdbmax = tdb;
   if (pdb > pdbmax) pdbmax = pdb;
   if (ethmag > ethmax) ethmax = ethmag;
   if (ephmag > ephmax) ephmax = ephmag;
   i++;
  }
  ipts = i;
  if (ithetas > maxthetas) maxthetas = ithetas;
  if (iphis > maxphis) maxphis = iphis;
  printf(" maxthetas: %d \n",maxthetas);
```

```
printf(" maxphis: %d \n",maxphis);
printf(" ethmax: %f \n",ethmax);
printf(" ephmax: %f \n",ephmax);
/* set aside memory */
tdbdata = (float ***)XtCalloc(maxphis, sizeof(float **));
pdbdata = (float ***)XtCalloc(maxphis, sizeof(float **));
ethdata = (float ***)XtCalloc(maxphis, sizeof(float **));
ephdata = (float ***)XtCalloc(maxphis, sizeof(float **));
ethphase = (float **)XtCalloc(maxphis, sizeof(float *));
ephphase = (float **)XtCalloc(maxphis, sizeof(float *));
for (i=0; i<maxphis; ++i)
{
 tdbdata[i] = (float **)XtCalloc(maxthetas, sizeof(float *));
 pdbdata[i] = (float **)XtCalloc(maxthetas, sizeof(float *));
 ethdata[i] = (float **)XtCalloc(maxthetas, sizeof(float *));
 ephdata[i] = (float **)XtCalloc(maxthetas, sizeof(float *));
 ethphase[i] = (float *)XtCalloc(maxthetas, sizeof(float));
 ephphase[i] = (float *)XtCalloc(maxthetas, sizeof(float));
 for (j=0; j<maxthetas; ++j)
 {
        tdbdata[i][j] = (float *)XtCalloc(3, sizeof(float));
        pdbdata[i][j] = (float *)XtCalloc(3, sizeof(float));
        ethdata[i][j] = (float *)XtCalloc(3, sizeof(float));
        ephdata[i][j] = (float *)XtCalloc(3, sizeof(float));
 }
}

/* rewind file */
rewind(infile_pt);
/* now go read data */
/* first look for frequency */
sprintf(lookfor1," FREQUENCY =%10.3f",selfreq);
fgets(line, MAXLINE, infile_pt);
*(line + 22) = '\0';
while(strcmp(lookfor1,line) != 0)
 {
        if (fgets(line, MAXLINE, infile_pt) == NULL) {
          printf("\n REACHED END of FILE!");
          exit(1);
        }
        *(line + 22) = '\0';
 }
/* now look for source */
sprintf(lookfor2," SOURCE :%5d",source);
fgets(line, MAXLINE, infile_pt);
*(line + 14) = '\0';
while(strcmp(lookfor2,line) != 0)
 {
        if (fgets(line, MAXLINE, infile_pt) == NULL) {
          printf("\n REACHED END of FILE!");
          exit(1);
        }
        *(line + 14) = '\0';
 }

/* dump next two lines */
fgets(line,MAXLINE,infile_pt);
fgets(line,MAXLINE,infile_pt);
iphis = 0;
ithetas = 0;
fscanf(infile_pt, "%f %f %f %f %f %f %f %f", &theta, &phi, &tdb,
          &pdb, &ethmag, &ethph, &ephmag, &ephph);
oldtheta = theta;
oldphi = phi;
tdb = 45.0 - (tdbmax - tdb);
pdb = 45.0 - (pdbmax - pdb);
if (tdb < 0.0) tdb = 1.0;
if (pdb < 0.0) pdb = 1.0;
if (ethmag <= 0.01 * ethmax) ethmag = 0.01 * ethmax;
if (ephmag <= 0.01 * ephmax) ephmag = 0.01 * ephmax;
tdbdata[iphis][ithetas][0] =2.0*tdb*cos(cdr*phi)*sin(cdr*theta)/45.0;
tdbdata[iphis][ithetas][1] =2.0*tdb*sin(cdr*phi)*sin(cdr*theta)/45.0;
tdbdata[iphis][ithetas][2] =2.0*tdb*cos(cdr*theta)/45.0;
pdbdata[iphis][ithetas][0] =2.0*pdb*cos(cdr*phi)*sin(cdr*theta)/45.0;
pdbdata[iphis][ithetas][1] =2.0*pdb*sin(cdr*phi)*sin(cdr*theta)/45.0;
pdbdata[iphis][ithetas][2] =2.0*pdb*cos(cdr*theta)/45.0;
ethdata[iphis][ithetas][0] =2.0*ethmag*cos(cdr*phi)*sin(cdr*theta)/ethmax;
ethdata[iphis][ithetas][1] =2.0*ethmag*sin(cdr*phi)*sin(cdr*theta)/ethmax;
ethdata[iphis][ithetas][2] =2.0*ethmag*cos(cdr*theta)/ethmax;
ephdata[iphis][ithetas][0] =2.0*ephmag*cos(cdr*phi)*sin(cdr*theta)/ephmax;
ephdata[iphis][ithetas][1] =2.0*ephmag*sin(cdr*phi)*sin(cdr*theta)/ephmax;
ephdata[iphis][ithetas][2] =2.0*ephmag*cos(cdr*theta)/ephmax;
/* put phase between 0 and 360 */
while (ethph < 0.0) ethph = ethph + 360;
while (ethph > 360.0) ethph = ethph - 360;
ethphase[iphis][ithetas] = ethph;
while (ephph < 0.0) ephph = ephph + 360;
while (ephph > 360.0) ephph = ephph - 360;
ephphase[iphis][ithetas] = ephph;
for (i=1; i < ipts; i++)
 {
        fscanf(infile_pt, "%f %f %f %f %f %f %f %f", &theta, &phi, &tdb,
            &pdb, &ethmag, &ethph, &ephmag, &ephph);
```

```
                if (theta > oldtheta) ithetas++;
                if (phi > oldphi) iphis++;
                if (theta < oldtheta) ithetas = 0;
                if (phi < oldphi) iphis = 0;
                tdb = 45.0 - (tdbmax - tdb);
                pdb = 45.0 - (pdbmax - pdb);
                if (tdb < 0.0) tdb = 1.0;
                if (pdb < 0.0) pdb = 1.0;
                if (ethmag <= 0.01 * ethmax) ethmag = 0.01 * ethmax;
                if (ephmag <= 0.01 * ephmax) ephmag = 0.01 * ephmax;
                tdbdata[iphis][ithetas][0] =2.0*tdb*cos(cdr*phi)*sin(cdr*theta)/45.0;
                tdbdata[iphis][ithetas][1] =2.0*tdb*sin(cdr*phi)*sin(cdr*theta)/45.0;
                tdbdata[iphis][ithetas][2] =2.0*tdb*cos(cdr*theta)/45.0;
                pdbdata[iphis][ithetas][0] =2.0*pdb*cos(cdr*phi)*sin(cdr*theta)/45.0;
                pdbdata[iphis][ithetas][1] =2.0*pdb*sin(cdr*phi)*sin(cdr*theta)/45.0;
                pdbdata[iphis][ithetas][2] =2.0*pdb*cos(cdr*theta)/45.0;
                ethdata[iphis][ithetas][0] =2.0*ethmag*cos(cdr*phi)*sin(cdr*theta)/ethmax;
                ethdata[iphis][ithetas][1] =2.0*ethmag*sin(cdr*phi)*sin(cdr*theta)/ethmax;
                ethdata[iphis][ithetas][2] =2.0*ethmag*cos(cdr*theta)/ethmax;
                ephdata[iphis][ithetas][0] =2.0*ephmag*cos(cdr*phi)*sin(cdr*theta)/ephmax;
                ephdata[iphis][ithetas][1] =2.0*ephmag*sin(cdr*phi)*sin(cdr*theta)/ephmax;
                ephdata[iphis][ithetas][2] =2.0*ephmag*cos(cdr*theta)/ephmax;
                while (ethph < 0.0) ethph = ethph + 360;
                while (ethph > 360.0) ethph = ethph - 360;
                ethphase[iphis][ithetas] = ethph;
                while (ephph < 0.0) ephph = ephph + 360;
                while (ephph > 360.0) ephph = ephph - 360;
                ephphase[iphis][ithetas] = ephph;
                oldtheta = theta;
                oldphi = phi;
    }
   fclose(infile_pt);
   axes[0].x = 0.0;
   axes[0].y = 0.0;
   axes[0].z = 0.0;
   axes[1].x = 0.0;
   axes[1].y = -2.5;
   axes[1].z = 0.0;
   axes[2].x = -2.5;
   axes[2].y = 0.0;
   axes[2].z = 2.5;
   axes[3].x = 0.0;
   axes[3].y = 0.0;
   axes[3].z = 2.5;
 }
}

/*****************************************************************************/

static void DrawCallback(Widget widget, XtPointer call_data,
                                        XmDrawingAreaCallbackStruct *cbs)
{
 XEvent   *event = cbs->event;
 XgPoint  wpoint;
 XgPoint  segpoint[2];
 XgNcPoint point;
 XgBoolean hit;
 int      segment,pick_id;
 XgOrient orient;
 XgZoom   zoom;
 XgExtents extents;
 Dimension width,height;
 DrawData *drawData;
 int  plttype;
 int wireNumber;
 int wind;
 char string [3000];
                char line [100];
 unsigned long werror;
 static Widget text;
 Widget       form;
 Arg          args [10];
 int          n;
 XmString     xmstring;
 extern Widget topLevel;
 Widget pane;
 void editWireButtonCB();
 void editNodeButtonCB();
 void closeButtonCB();
 static ActionAreaItem actionItems[] = {
   {"Edit Wire", editWireButtonCB, NULL},
   {"Edit Node1", editNodeButtonCB, (XtPointer)1},
   {"Edit Node2", editNodeButtonCB, (XtPointer)2},
   {"Close", closeButtonCB, NULL}
 };
 extern void newEscapeAction();

 XtVaGetValues (widget, XmNuserData, &drawData, NULL);
 XgSetCurrentDevice (drawData->xgd);

 XgInquireOrthographicView(&orient,&zoom,&extents);
 XtVaSetValues(drawData->az_scale, XmNvalue,(int)orient.az, NULL);
 XtVaSetValues(drawData->el_scale, XmNvalue,(int)orient.el, NULL);
```

```
  XtVaSetValues(drawData->x_scale, XmNvalue,(int)(zoom.xcenter*100), NULL);
  XtVaSetValues(drawData->y_scale, XmNvalue,(int)(zoom.ycenter*100), NULL);
  XtVaSetValues(drawData->zoom_scale, XmNvalue,(int)(zoom.scale*100), NULL);

  switch(event->type) {
  case ButtonPress:
/* this section will be modified to bring up wire/node edit window */
   plttype = drawData->type;
   XtVaGetValues(widget,XmNwidth,&width,XmNheight,&height,NULL);
   height--;
   width--;
   point.x = (float)event->xbutton.x / (float)width;
   point.y = (float)(height - event->xbutton.y) / (float)height;
   XgConvertNcToWc(&point,&wpoint);
   XgDeleteSegment(500);
   XgRedrawAllSegments();
   hit = XgLocateSegment(&point,&segment,&pick_id);
   if ((hit == 1)&&(segment != 0)&&(segment != 999)&&(segment != 998)) {

/* put a marker at this spot */
    XgOpenSegment(500);
    XgSetPolymarkerSize(2.0);
    XgSetPolymarkerColor(XG_WHITE);
    XgSetPolymarkerType(XG_STAR);
    segpoint[0].x = bgn_points[pick_id].x;
    segpoint[0].y = bgn_points[pick_id].y;
    segpoint[0].z = bgn_points[pick_id].z;
    segpoint[1].x = end_points[pick_id].x;
    segpoint[1].y = end_points[pick_id].y;
    segpoint[1].z = end_points[pick_id].z;
    XgPolymarker(2,segpoint);
    XgCloseSegment();
    XgRedrawAllSegments();

    wireNumber = Jwire[pick_id];
    sprintf(line,"**********************************************************************\n");
    strcpy(string,line);
    sprintf(line,"You just selected SEGMENT number %d\n",pick_id+1);
    strcat(string,line);
    if (wireNumber <= SWireCount) {
/* this section is for straight wires */
            wind = wireNumber-1;
            sprintf(line,"This is SEGMENT number %d on STRAIGHT WIRE number %d\n",
               Jseg[pick_id],wireNumber);
            strcat(string,line);
            sprintf(line,"  End #1 of this wire is NODE %d (%f,%f,%f).\n",
               GW_END1[wind],X[GW_END1[wind]-1],
               Y[GW_END1[wind]-1],Z[GW_END1[wind]-1]);
            strcat(string,line);
            sprintf(line,"  End #2 of this wire is NODE %d (%f,%f,%f).\n",
               GW_END2[wind],X[GW_END2[wind]-1],
               Y[GW_END2[wind]-1],Z[GW_END2[wind]-1]);
            strcat(string,line);
            sprintf(line,"  Wire RADIUS is %f meters.\n",GW_RAD[wind]);
            strcat(string,line);
            sprintf(line,"  Number of SEGMENTS is %d.\n",GW_NS[wind]);
            strcat(string,line);
    }
    else {
            wind = wireNumber-SWireCount-1;
            sprintf(line,"This is SEGMENT number %d on TAPERED WIRE number %d\n",
               Jseg[pick_id],wireNumber-SWireCount);
            strcat(string,line);
            sprintf(line,"  End #1 of this wire is NODE %d (%f,%f,%f).\n",
               GC_END1[wind],X[GC_END1[wind]-1],
               Y[GC_END1[wind]-1],Z[GC_END1[wind]-1]);
            strcat(string,line);
            sprintf(line,"  End #2 of this wire is NODE %d (%f,%f,%f).\n",
               GC_END2[wind],X[GC_END2[wind]-1],
               Y[GC_END2[wind]-1],Z[GC_END2[wind]-1]);
            strcat(string,line);
            sprintf(line,"  BEGINNING wire RADIUS is %f meters.\n",GC_RAD1[wind]);
            strcat(string,line);
            sprintf(line,"  ENDING wire RADIUS is %f meters.\n",GC_RAD2[wind]);
            strcat(string,line);
            sprintf(line,"  Number of SEGMENTS is %d.\n",GC_NS[wind]);
            strcat(string,line);
    }

    if (plttype == 0) {
    /* decode errors and warnings for diagnostics*/
            sprintf(line,"Results of DIAGNOSTICS for this wire are: \n");
            strcat(string,line);
            werror = wireErrors[wireNumber-1];
            if (werror == 0) {
             sprintf(line,"  No ERRORS or WARNINGS!\n");
             strcat(string,line);
            } else {
             if ((werror & SegLen2WaveLenWarning) == SegLen2WaveLenWarning) {
              sprintf(line,"  Segment Length To Wavelength WARNING! \n");
              strcat(string,line); }
             if ((werror & SegLen2RadiusWarning) == SegLen2RadiusWarning) {
              sprintf(line,"  Segment Length To Radius WARNING! \n");
```

```
                strcat(string,line); }
              if ((werror & Radius2WaveLenWarning) == Radius2WaveLenWarning) {
                sprintf(line," Radius To Wavelength WARNING! \n");
                strcat(string,line); }
              if ((werror & JunctionSegLenRatioWarning) == JunctionSegLenRatioWarning) {
                sprintf(line," Junction Segment Length Ratio WARNING! \n");
                strcat(string,line); }
              if ((werror & JunctionRadiusRatioWarning) == JunctionRadiusRatioWarning) {
                sprintf(line," Junction Radius Ratio WARNING! \n");
                strcat(string,line); }
              if ((werror & SegLen2WaveLenError) == SegLen2WaveLenError) {
                sprintf(line," Segment Length To Wavelength ERROR! \n");
                strcat(string,line); }
              if ((werror & SegLen2RadiusError) == SegLen2RadiusError) {
                sprintf(line," Segment Length To Radius ERROR! \n");
                strcat(string,line); }
              if ((werror & Radius2WaveLenError) == Radius2WaveLenError) {
                sprintf(line," Radius To Wavelength ERROR! \n");
                strcat(string,line); }
              if ((werror & CoincidentWireError) == CoincidentWireError) {
                sprintf(line," Coincindent Wire ERROR! \n");
                strcat(string,line); }
              if ((werror & JunctionSegLenRatioError) == JunctionSegLenRatioError) {
                sprintf(line," Junction Segment Length Ratio ERROR! \n");
                strcat(string,line); }
              if ((werror & JunctionRadiusRatioError) == JunctionRadiusRatioError) {
                sprintf(line," Junction Radius Ratio ERROR! \n");
                strcat(string,line); }
              if ((werror & JunctionMatchPointError) == JunctionMatchPointError) {
                sprintf(line," Junction Match Point ERROR! \n");
                strcat(string,line); }
              if ((werror & CrossedWireError) == CrossedWireError) {
                sprintf(line," Crossed Wire ERROR! \n");
                strcat(string,line); }
              if ((werror & InvalidSheathRadiusError) == InvalidSheathRadiusError) {
                sprintf(line," Invalid Sheath Radius ERROR! \n");
                strcat(string,line); }
            }
    }
    xmstring = XmStringCreateLtoR (string, XmSTRING_DEFAULT_CHARSET);

    if (popupShell == NULL) {
            n = 0;
            XtSetArg (args [n], XmNx, 50); n++;
            XtSetArg (args [n], XmNy, 50); n++;
            XtSetArg (args [n], XmNheight, 200); n++;
            XtSetArg (args [n], XmNwidth, 600); n++;
            popupShell = XtCreatePopupShell ("Pick Information",
              topLevelShellWidgetClass, topLevel, args, n);
            newEscapeAction(popupShell);
            XtSetArg (args [0], XmNsashWidth, 1);
            XtSetArg (args [1], XmNsashHeight, 1);
            pane = XmCreatePanedWindow (popupShell, "pane", args, 2);
            XtManageChild (pane);
            form = XmCreateForm (pane, "form", NULL, 0);
            n = 0;
            XtSetArg (args [n], XmNtopAttachment, XmATTACH_FORM); n++;
            XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
            XtSetArg (args [n], XmNleftOffset, 5); n++;
            XtSetArg (args [n], XmNrightAttachment, XmATTACH_FORM); n++;
            XtSetArg (args [n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
            XtSetArg (args [n], XmNlabelString, xmstring); n++;
            text = XmCreateLabel(pane, "label", args, n);

            XtManageChild(text);
            XtManageChild (form);
            actionItems[3].data = (XtPointer) popupShell;
            createActionArea (pane, actionItems, 4);

            XtManageChild(popupShell);
    } else {
            XtSetArg (args [0], XmNlabelString, xmstring);
            XtSetValues (text, args, 1);
    }
    XmStringFree (xmstring);
    XtPopup (popupShell, XtGrabNone);       /* Popup the window */
    wireIndex = wireNumber;
   }
   else {
    if (popupShell)
            XtPopdown(popupShell);
    popupShell = NULL;
   }
   break;
  }
  (void)widget;
  (void)call_data;
}

/*****************************************************************************/

static void PushCallback(Widget widget, int tag,
                                XmAnyCallbackStruct *reason)
```

```
{
 XgDevice ps_xgd;
 DrawData *drawData;
 int     plttype;
 static XgVisible visible = XG_VISIBLE;

 XtVaGetValues (XtParent (widget), XmNuserData, &drawData, NULL);
 XgSetCurrentDevice (drawData->xgd);
 plttype = drawData->type;

 switch(tag) {
 case 0:                 /* Toggle Axes */
  visible = (visible == XG_VISIBLE) ? XG_INVISIBLE : XG_VISIBLE;
  XgSetSegmentVisibility(999,visible);
  XgRedrawAllSegments();
  break;
 case 1:                 /* Reset */
  Reset (drawData);
  break;
 case 2:{                /* Print */
  char command[80], tmpfile[40];

  tmpnam (tmpfile);
  ps_xgd = XgOpenPostscript (tmpfile, XG_PORTRAIT);
  XgAssociateDevice(ps_xgd,drawData->xgd,True);
  XgCloseDevice(ps_xgd);
  XgSetCurrentDevice(drawData->xgd);
  sprintf (command, "%s %s", getenv ("MOM_PRINT_GRAPHICS"), tmpfile);
  system (command);
  remove (tmpfile);
  break;
 }
 case 3:                 /* quit */
  XtDestroyWidget (XtParent (XtParent (XtParent (widget))));
  break;
 default:
  printf("tag=%d",tag);
  (void)widget;
  (void)reason;
  break;
 }
}

/*****************************************************************************/

static void PushCallback2 (Widget widget, int tag,
                           XmAnyCallbackStruct *reason)
{
 XgOrient orient;
 XgZoom zoom;
 XgExtents extents;
 DrawData *drawData;

 /* Get Xgraphics device from widget parent & make it current */
 XtVaGetValues (XtParent (XtParent (widget)), XmNuserData, &drawData, NULL);
 XgSetCurrentDevice (drawData->xgd);

 XgInquireOrthographicView(&orient,&zoom,&extents);

 switch(tag) {
 case 0:                 /* Plan */
  orient.az = 0;
  orient.el = 0;
  break;
 case 1:                 /* Elevation */
  orient.az = 0;
  orient.el = 90;
  break;
 case 2:                 /* Bow */
  orient.az = 90;
  orient.el = 0;
  break;
 default:
  printf("tag=%d",tag);
  (void)widget;
  (void)reason;
  break;
 }
 zoom.xcenter = 0.5;
 zoom.ycenter = 0.5;
 zoom.scale = 1.0;
 XgSetOrthographicView(&orient,&zoom,&extents);
 XtVaSetValues(drawData->az_scale, XmNvalue,(int)orient.az, NULL);
 XtVaSetValues(drawData->el_scale, XmNvalue,(int)orient.el, NULL);
 XtVaSetValues(drawData->x_scale, XmNvalue,(int)(zoom.xcenter*100), NULL);
 XtVaSetValues(drawData->y_scale, XmNvalue,(int)(zoom.ycenter*100), NULL);
 XtVaSetValues(drawData->zoom_scale, XmNvalue,(int)(zoom.scale*100), NULL);
 (void)widget;
}

/*****************************************************************************/

static void ScaleCallback(Widget widget, int tag,
```

```
                                        XmScaleCallbackStruct *cbs)
{
 XgOrient  orient;
 XgZoom    zoom;
 XgExtents extents;
 DrawData  *drawData;

 /* Get Xgraphics device from widget parent & make it current */
 XtVaGetValues (XtParent (widget), XmNuserData, &drawData, NULL);
 XgSetCurrentDevice (drawData->xgd);

 XgInquireOrthographicView(&orient,&zoom,&extents);

 switch(tag) {
 case 0:
  /*
   printf("az=%d\n",cbs->value);
   */
  orient.az = cbs->value;
  break;
 case 1:
  /*
   printf("el=%d\n",cbs->value);
   */
  orient.el = cbs->value;
  break;
 case 2:
  /*
   printf("x=%d\n",cbs->value);
   */
  zoom.xcenter = (float)cbs->value / 100.0;
  break;
 case 3:
  /*
   printf("y=%d\n",cbs->value);
   */
  zoom.ycenter = (float)cbs->value / 100.0;
  break;
 case 4:
  /*
   printf("scale=%d\n",cbs->value);
   */
  zoom.scale = (float)cbs->value / 100.0;
  break;
 }
 XgSetOrthographicView(&orient,&zoom,&extents);
 (void)widget;
}

/*****************************************************************************/

int Reset (DrawData *drawData)
{
 XgOrient orient;
 XgZoom zoom;
 XgExtents extents;

 XgInquireOrthographicView(&orient,&zoom,&extents);
 orient.az = 0;
 orient.el = 0;
 zoom.xcenter = 0.5;
 zoom.ycenter = 0.5;
 zoom.scale = 1.0;
 XgSetOrthographicView(&orient,&zoom,&extents);
 XtVaSetValues(drawData->az_scale, XmNvalue,(int)orient.az, NULL);
 XtVaSetValues(drawData->el_scale, XmNvalue,(int)orient.el, NULL);
 XtVaSetValues(drawData->x_scale, XmNvalue,(int)(zoom.xcenter*100), NULL);
 XtVaSetValues(drawData->y_scale, XmNvalue,(int)(zoom.ycenter*100), NULL);
 XtVaSetValues(drawData->zoom_scale,XmNvalue,(int)(zoom.scale*100), NULL);

 return 0;
}

/*****************************************************************************/

int DrawNE(plttype)
    int plttype;
{
 int i;
 float val;

 XgOpenSegment(998);
 XgSetSegmentDetectability(998,XG_DETECTABLE);
 for (i=0; i<numnepts; i++) {
  XgSetPickId(i);
  XgSetPolymarkerType(XG_POINT);
  if (plttype == 13) val = ne_val[0][i];
  if (plttype == 14) val = ne_val[1][i];
  if (val < 1.0) XgSetPolymarkerColor(XG_MAGENTA);
  else if (val < 2.0) XgSetPolymarkerColor(XG_BLUE);
  else if (val < 3.0) XgSetPolymarkerColor(XG_CYAN);
  else if (val < 4.0) XgSetPolymarkerColor(XG_GREEN);
  else if (val < 5.0) XgSetPolymarkerColor(XG_YELLOW);
```

```
    else if (val < 6.0) XgSetPolymarkerColor(XG_RED);
    else XgSetPolymarkerColor(XG_WHITE);
    XgPolymarker(1,&ne_points[i]);
  }
  XgCloseSegment();
  /* Free up memory */
  for (i=0; i<2; ++i)
  {
    XtFree((char *) ne_val[i]);
  }
  XtFree((char *) ne_val);
  XtFree((char *) ne_points);
  return 0;
}
/*****************************************************************************/

int DrawNH(plttype)
    int plttype;
{
  int i;
  float val;

  XgOpenSegment(997);
  XgSetSegmentDetectability(997,XG_DETECTABLE);
  for (i=0; i<numnhpts; i++) {
    XgSetPickId(i);
    XgSetPolymarkerType(XG_POINT);
    if (plttype == 15) val = nh_val[0][i];
    if (plttype == 16) val = nh_val[1][i];
    if (val < 1.0) XgSetPolymarkerColor(XG_MAGENTA);
    else if (val < 2.0) XgSetPolymarkerColor(XG_BLUE);
    else if (val < 3.0) XgSetPolymarkerColor(XG_CYAN);
    else if (val < 4.0) XgSetPolymarkerColor(XG_GREEN);
    else if (val < 5.0) XgSetPolymarkerColor(XG_YELLOW);
    else if (val < 6.0) XgSetPolymarkerColor(XG_RED);
    else XgSetPolymarkerColor(XG_WHITE);
    XgPolymarker(1,&nh_points[i]);
  }
  XgCloseSegment();
  /* Free up memory */
  for (i=0; i<2; ++i)
  {
    XtFree((char *) nh_val[i]);
  }
  XtFree((char *) nh_val);
  XtFree((char *) nh_points);
  return 0;
}
/*****************************************************************************/

int DrawGeo (int plttype)
{
  int i;
  XgPoint points[2];

  /* Draw Axes */
  XgOpenSegment(999);
  XgSetSegmentDetectability(999,XG_DETECTABLE);
  XgSetPolylineColor(XG_RED);
  memcpy(&points[0],&axes[0],sizeof(XgPoint));
  memcpy(&points[1],&axes[1],sizeof(XgPoint));
  XgPolyline(2,points);
  XgSetPolylineColor(XG_GREEN);
  memcpy(&points[0],&axes[0],sizeof(XgPoint));
  memcpy(&points[1],&axes[2],sizeof(XgPoint));
  XgPolyline(2,points);
  XgSetPolylineColor(XG_BLUE);
  memcpy(&points[0],&axes[0],sizeof(XgPoint));
  memcpy(&points[1],&axes[3],sizeof(XgPoint));
  XgPolyline(2,points);
  XgCloseSegment();

  XgOpenSegment(plttype+1);
  XgSetSegmentDetectability(plttype+1,XG_DETECTABLE);
  for (i=0; i<numsegs; i++) {
    memcpy(&points[0],&bgn_points[i],sizeof(XgPoint));
    memcpy(&points[1],&end_points[i],sizeof(XgPoint));
    XgSetPickId(i);
    switch(ival[plttype][i]) {
    case 0: XgSetPolylineColor(XG_MAGENTA);
      break;
    case 1: XgSetPolylineColor(XG_BLUE);
      break;
    case 2: XgSetPolylineColor(XG_CYAN);
      break;
    case 3: XgSetPolylineColor(XG_GREEN);
      break;
    case 4: XgSetPolylineColor(XG_YELLOW);
      break;
    case 5: XgSetPolylineColor(XG_RED);
      break;
    case 6: XgSetPolylineColor(XG_WHITE);
      break;
```

```
    }
    XgPolyline(2,points);
  }
  ColorKey (plttype);
  XgCloseSegment();
  /* Free up memory */
  for (i=0; i<17; ++i)
  {
    XtFree((char *) ival[i]);
  }
  XtFree((char *) ival);
/*
  XtFree((char *) bgn_points);
  XtFree((char *) end_points);
*/

  return 0;
}

/*****************************************************************************/

int DrawPat (int plttype)
{
  int ithetas, iphis;
              int i,j;
  XgPoint points[2];
  XgPoint pt_verts[4];

  /* Draw Axes */
  XgOpenSegment(999);
  XgSetSegmentDetectability(999,XG_DETECTABLE);
  XgSetPolylineColor(XG_RED);
  memcpy(&points[0],&axes[0],sizeof(XgPoint));
  memcpy(&points[1],&axes[1],sizeof(XgPoint));
  XgPolyline(2,points);
  XgSetPolylineColor(XG_GREEN);
  memcpy(&points[0],&axes[0],sizeof(XgPoint));
  memcpy(&points[1],&axes[2],sizeof(XgPoint));
  XgPolyline(2,points);
  XgSetPolylineColor(XG_BLUE);
  memcpy(&points[0],&axes[0],sizeof(XgPoint));
  memcpy(&points[1],&axes[3],sizeof(XgPoint));
  XgPolyline(2,points);
  XgCloseSegment();

  XgOpenSegment(plttype+1);
  XgSetSegmentDetectability(plttype+1,XG_DETECTABLE);
  for (ithetas = 0; ithetas < maxthetas-1; ithetas++) {
    for (iphis = 0; iphis < maxphis-1; iphis++) {
      XgSetPickId(ithetas*maxphis+iphis);
      switch (plttype) {
      case 17:
                pt_verts[0].x = -ethdata[iphis][ithetas][1];
                pt_verts[0].y = -ethdata[iphis][ithetas][0];
                pt_verts[0].z = ethdata[iphis][ithetas][2];
                pt_verts[1].x = -ethdata[iphis][ithetas+1][1];
                pt_verts[1].y = -ethdata[iphis][ithetas+1][0];
                pt_verts[1].z = ethdata[iphis][ithetas+1][2];
                if (iphis == maxphis-1) {
                  pt_verts[2].x = -ethdata[0][ithetas+1][1];
                  pt_verts[2].y = -ethdata[0][ithetas+1][0];
                  pt_verts[2].z = ethdata[0][ithetas+1][2];
                  pt_verts[3].x = -ethdata[0][ithetas][1];
                  pt_verts[3].y = -ethdata[0][ithetas][0];
                  pt_verts[3].z = ethdata[0][ithetas][2];
                  phase = (ethphase[iphis][ithetas]+
                                ethphase[iphis][ithetas+1]+
                                ethphase[0][ithetas]+
                                ethphase[0][ithetas+1])/4.0;
                } else {
                  pt_verts[2].x = -ethdata[iphis+1][ithetas+1][1];
                  pt_verts[2].y = -ethdata[iphis+1][ithetas+1][0];
                  pt_verts[2].z = ethdata[iphis+1][ithetas+1][2];
                  pt_verts[3].x = -ethdata[iphis+1][ithetas][1];
                  pt_verts[3].y = -ethdata[iphis+1][ithetas][0];
                  pt_verts[3].z = ethdata[iphis+1][ithetas][2];
                  phase = (ethphase[iphis][ithetas]+
                                ethphase[iphis][ithetas+1]+
                                ethphase[iphis+1][ithetas]+
                                ethphase[iphis+1][ithetas+1])/4.0;
                }
                break;
      case 18:
                pt_verts[0].x = -tdbdata[iphis][ithetas][1];
                pt_verts[0].y = -tdbdata[iphis][ithetas][0];
                pt_verts[0].z = tdbdata[iphis][ithetas][2];
                pt_verts[1].x = -tdbdata[iphis][ithetas+1][1];
                pt_verts[1].y = -tdbdata[iphis][ithetas+1][0];
                pt_verts[1].z = tdbdata[iphis][ithetas+1][2];
                if (iphis == maxphis-1) {
                  pt_verts[2].x = -tdbdata[0][ithetas+1][1];
                  pt_verts[2].y = -tdbdata[0][ithetas+1][0];
                  pt_verts[2].z = tdbdata[0][ithetas+1][2];
```

```
                pt_verts[3].x = -tdbdata[0][ithetas][1];
                pt_verts[3].y = -tdbdata[0][ithetas][0];
                pt_verts[3].z = tdbdata[0][ithetas][2];
                phase = (ethphase[iphis][ithetas]+
                        ethphase[iphis][ithetas+1]+
                        ethphase[0][ithetas]+
                        ethphase[0][ithetas+1])/4.0;
              } else {
                pt_verts[2].x = -tdbdata[iphis+1][ithetas+1][1];
                pt_verts[2].y = -tdbdata[iphis+1][ithetas+1][0];
                pt_verts[2].z = tdbdata[iphis+1][ithetas+1][2];
                pt_verts[3].x = -tdbdata[iphis+1][ithetas][1];
                pt_verts[3].y = -tdbdata[iphis+1][ithetas][0];
                pt_verts[3].z = tdbdata[iphis+1][ithetas][2];
                phase = (ethphase[iphis][ithetas]+
                        ethphase[iphis][ithetas+1]+
                        ethphase[iphis+1][ithetas]+
                        ethphase[iphis+1][ithetas+1])/4.0;
              }
              break;
case 19:
              pt_verts[0].x = -ephdata[iphis][ithetas][1];
              pt_verts[0].y = -ephdata[iphis][ithetas][0];
              pt_verts[0].z = ephdata[iphis][ithetas][2];
              pt_verts[1].x = -ephdata[iphis][ithetas+1][1];
              pt_verts[1].y = -ephdata[iphis][ithetas+1][0];
              pt_verts[1].z = ephdata[iphis][ithetas+1][2];
              if (iphis == maxphis-1) {
                pt_verts[2].x = -ephdata[0][ithetas+1][1];
                pt_verts[2].y = -ephdata[0][ithetas+1][0];
                pt_verts[2].z = ephdata[0][ithetas+1][2];
                pt_verts[3].x = -ephdata[0][ithetas][1];
                pt_verts[3].y = -ephdata[0][ithetas][0];
                pt_verts[3].z = ephdata[0][ithetas][2];
                phase = (ephphase[iphis][ithetas]+
                        ephphase[iphis][ithetas+1]+
                        ephphase[0][ithetas]+
                        ephphase[0][ithetas+1])/4.0;
              } else {
                pt_verts[2].x = -ephdata[iphis+1][ithetas+1][1];
                pt_verts[2].y = -ephdata[iphis+1][ithetas+1][0];
                pt_verts[2].z = ephdata[iphis+1][ithetas+1][2];
                pt_verts[3].x = -ephdata[iphis+1][ithetas][1];
                pt_verts[3].y = -ephdata[iphis+1][ithetas][0];
                pt_verts[3].z = ephdata[iphis+1][ithetas][2];
                phase = (ephphase[iphis][ithetas]+
                        ephphase[iphis][ithetas+1]+
                        ephphase[iphis+1][ithetas]+
                        ephphase[iphis+1][ithetas+1])/4.0;
              }
              break;
case 20:
              pt_verts[0].x = -pdbdata[iphis][ithetas][1];
              pt_verts[0].y = -pdbdata[iphis][ithetas][0];
              pt_verts[0].z = pdbdata[iphis][ithetas][2];
              pt_verts[1].x = -pdbdata[iphis][ithetas+1][1];
              pt_verts[1].y = -pdbdata[iphis][ithetas+1][0];
              pt_verts[1].z = pdbdata[iphis][ithetas+1][2];
              if (iphis == maxphis-1) {
                pt_verts[2].x = -pdbdata[0][ithetas+1][1];
                pt_verts[2].y = -pdbdata[0][ithetas+1][0];
                pt_verts[2].z = pdbdata[0][ithetas+1][2];
                pt_verts[3].x = -pdbdata[0][ithetas][1];
                pt_verts[3].y = -pdbdata[0][ithetas][0];
                pt_verts[3].z = pdbdata[0][ithetas][2];
                phase = (ephphase[iphis][ithetas]+
                        ephphase[iphis][ithetas+1]+
                        ephphase[0][ithetas]+
                        ephphase[0][ithetas+1])/4.0;
              } else {
                pt_verts[2].x = -pdbdata[iphis+1][ithetas+1][1];
                pt_verts[2].y = -pdbdata[iphis+1][ithetas+1][0];
                pt_verts[2].z = pdbdata[iphis+1][ithetas+1][2];
                pt_verts[3].x = -pdbdata[iphis+1][ithetas][1];
                pt_verts[3].y = -pdbdata[iphis+1][ithetas][0];
                pt_verts[3].z = pdbdata[iphis+1][ithetas][2];
                phase = (ephphase[iphis][ithetas]+
                        ephphase[iphis][ithetas+1]+
                        ephphase[iphis+1][ithetas]+
                        ephphase[iphis+1][ithetas+1])/4.0;
              }
              break;
}
/* now put the phase between 0 and 360 */
while (phase < 0.0) phase = phase + 360;
while (phase > 360.0) phase = phase - 360;
/* now set the color based on the phase */
if (phase < 60.0) XgSetFillAreaInteriorColor(XG_MAGENTA);
else if (phase < 120.0) XgSetFillAreaInteriorColor(XG_BLUE);
else if (phase < 180.0) XgSetFillAreaInteriorColor(XG_CYAN);
else if (phase < 240.0) XgSetFillAreaInteriorColor(XG_GREEN);
else if (phase < 300.0) XgSetFillAreaInteriorColor(XG_YELLOW);
else XgSetFillAreaInteriorColor(XG_RED);
```

```
      /* draw the polyhedron */
      XgSetFillAreaInteriorType(XG_FILLED);
      XgFillArea(4, pt_verts);
    }
  }
  ColorKey (plttype);
  XgCloseSegment();
  /* Free up memory */
  for (i=0; i<maxphis; ++i)
  {
    for (j=0; j<maxthetas; ++j)
    {
      XtFree ((char *) tdbdata[i][j]);
      XtFree ((char *) pdbdata[i][j]);
      XtFree ((char *) ethdata[i][j]);
      XtFree ((char *) ephdata[i][j]);
    }
    XtFree ((char *) tdbdata[i]);
    XtFree ((char *) pdbdata[i]);
    XtFree ((char *) ethdata[i]);
    XtFree ((char *) ephdata[i]);
    XtFree ((char *) ethphase[i]);
    XtFree ((char *) ephphase[i]);
  }
  XtFree ((char *) tdbdata);
  XtFree ((char *) pdbdata);
  XtFree ((char *) ethdata);
  XtFree ((char *) ephdata);
  XtFree ((char *) ethphase);
  XtFree ((char *) ephphase);


  return 0;
}

/*****************************************************************************/

int ColorKey (int plttype)
{
  char keystring[20];
  float val[6];
  float range,minv,maxv;

  XgSetCharacterHeight (10.0);
  if (plttype == 0) {
    XgSetTextColor(XG_WHITE);
    XgWindowText( -4.5, 1.0, "WIRE DIAGNOSTICS");
    XgWindowText( -3.0, 7.0, "No Problems");
    XgWindowText( -2.0, 7.0, "WARNING MESSAGE");
    XgWindowText( -1.0, 7.0, "ERROR MESSAGE");
    XgSetCharacterHeight ( 5.0);
    XgSetTextColor(XG_WHITE);
    XgWindowText( -5.0, 1.0, " ------");
    XgSetTextColor(XG_YELLOW);
    XgWindowText( -3.0, 1.0, " ------");
    XgSetTextColor(XG_RED);
    XgWindowText( -1.0, 1.0, " ------");
    XgSetCharacterHeight (10.0);
  }
  if (plttype == 4) {
    XgSetTextColor(XG_WHITE);
    XgWindowText( -4.0, 1.0, "WIRE CONNECTIONS");
    XgWindowText( -3.0, 7.0, "Both end connected");
    XgWindowText( -2.0, 7.0, "One end connected");
    XgWindowText( -1.0, 7.0, "Neither end connected");
    XgSetCharacterHeight ( 5.0);
    XgSetTextColor(XG_MAGENTA);
    XgWindowText( -5.0, 1.0, " ------");
    XgSetTextColor(XG_CYAN);
    XgWindowText( -3.0, 1.0, " ------");
    XgSetTextColor(XG_YELLOW);
    XgWindowText( -1.0, 1.0, " ------");
    XgSetCharacterHeight (10.0);
  }
  if ((plttype != 0)&&(plttype != 4)&&(plttype != 10)&&(plttype < 17))
    {
      maxv = maxval[plttype-1];
      minv = minval[plttype-1];
      range = maxv-minv;
      if ((plttype == 5)||(plttype == 6)||(plttype == 7)) {
                val[0] = minv/(pow(10.,log10(minv/maxv)/7));
                val[1] = minv/(pow(10.,2*log10(minv/maxv)/7));
                val[2] = minv/(pow(10.,3*log10(minv/maxv)/7));
                val[3] = minv/(pow(10.,4*log10(minv/maxv)/7));
                val[4] = minv/(pow(10.,5*log10(minv/maxv)/7));
                val[5] = minv/(pow(10.,6*log10(minv/maxv)/7));
      }
      else {
                val[0] = minv+range/7;
                val[1] = minv+2*range/7;
                val[2] = minv+3*range/7;
                val[3] = minv+4*range/7;
```

```
            val[4] = minv+5*range/7;
            val[5] = minv+6*range/7;
        }
        XgSetTextColor(XG_WHITE);
        switch (plttype) {
        case 1: XgWindowText( -8.0, 1.0, "SEGMENT LENGTH, METERS (LINEAR SCALE)");
                break;
        case 2: XgWindowText( -8.0, 1.0, "WIRE RADIUS, METERS (LINEAR SCALE)");
                break;
        case 3: XgWindowText( -8.0, 1.0, "SEGMENT TO RADIUS RATIO (LINEAR SCALE)");
                break;
        case 5: XgWindowText( -8.0, 1.0, "SEGMENT LENGTH, METERS (LOG SCALE)");
                break;
        case 6: XgWindowText( -8.0, 1.0, "WIRE RADIUS, METERS (LOG SCALE)");
                break;
        case 7: XgWindowText( -8.0, 1.0, "SEGMENT TO RADIUS RATIO (LOG SCALE)");
                break;
        case 8: XgWindowText( -8.0, 1.0, "CURRENT MAGNITUDE, AMPS (LINEAR SCALE)");
                break;
        case 9: XgWindowText( -8.0, 1.0, "CURRENT MAGNITUDE, AMPS (LOG SCALE)");
                break;
        case 11: XgWindowText( -8.0, 1.0, "CHARGE MAGNITUDE, COULOMBS (LINEAR SCALE)");
                break;
        case 12: XgWindowText( -8.0, 1.0, "CHARGE MAGNITUDE, COULOMBS (LOG SCALE)");
                break;
        case 13: XgWindowText( -8.0, 1.0, "Z-COMPONENT OF E-NORMAL, VOLTS/METER");
                break;
        case 14: XgWindowText( -8.0, 1.0, "TOTAL E-NORMAL, VOLTS/METER");
                break;
        case 15: XgWindowText( -8.0, 1.0, "X-COMPONENT OF H-NORMAL");
                break;
        case 16: XgWindowText( -8.0, 1.0, "Y-COMPONENT OF H-NORMAL");
                break;
        }
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_MAGENTA);
        XgWindowText( -13.0, 1.0, " ------");
        sprintf(keystring,"%5.2f - %5.2f",minv,val[0]);
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -7.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_BLUE);
        XgWindowText( -11.0, 1.0, " ------");
        sprintf(keystring,"%5.2f - %5.2f",val[0],val[1]);
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -6.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_CYAN);
        XgWindowText( -9.0, 1.0, " ------");
        sprintf(keystring,"%5.2f - %5.2f",val[1],val[2]);
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -5.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_GREEN);
        XgWindowText( -7.0, 1.0, " ------");
        sprintf(keystring,"%5.2f - %5.2f",val[2],val[3]);
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -4.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_YELLOW);
        XgWindowText( -5.0, 1.0, " ------");
        sprintf(keystring,"%5.2f - %5.2f",val[3],val[4]);
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -3.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_RED);
        XgWindowText( -3.0, 1.0, " ------");
        sprintf(keystring,"%5.2f - %5.2f",val[4],val[5]);
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -2.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -1.0, 1.0, " ------");
        sprintf(keystring,"%5.2f - %5.2f",val[5],maxv);
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -1.0, 6.0, keystring);
    }
    if ((plttype == 10)||(plttype >= 17))
    {
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        switch (plttype) {
        case 10: XgWindowText( -8.0, 1.0, "Current, PHASE");
                break;
        case 17: XgWindowText( -8.0, 1.0, "E-THETA, PHASE");
                break;
```

```
        case 18: XgWindowText( -8.0, 1.0, "E-THETA, DB, PHASE");
                 break;
        case 19: XgWindowText( -8.0, 1.0, "E-PHI, PHASE");
                 break;
        case 20: XgWindowText( -8.0, 1.0, "E-PHI, DB, PHASE");
                 break;
        }
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_MAGENTA);
        XgWindowText( -13.0, 1.0, " ------");
        sprintf(keystring,"0 - 60");
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -7.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_BLUE);
        XgWindowText( -11.0, 1.0, " ------");
        sprintf(keystring,"60 - 120");
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -8.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_CYAN);
        XgWindowText( -9.0, 1.0, " ------");
        sprintf(keystring,"120 - 180");
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -5.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_GREEN);
        XgWindowText( -7.0, 1.0, " ------");
        sprintf(keystring,"180 - 240");
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -4.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_YELLOW);
        XgWindowText( -5.0, 1.0, " ------");
        sprintf(keystring,"240 - 300");
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -3.0, 6.0, keystring);
        XgSetCharacterHeight (5.0);
        XgSetTextColor(XG_RED);
        XgWindowText( -3.0, 1.0, " ------");
        sprintf(keystring,"300 - 360");
        XgSetCharacterHeight (10.0);
        XgSetTextColor(XG_WHITE);
        XgWindowText( -2.0, 6.0, keystring);
      }
      return 0;
}

/*****************************************************************/
void closeButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  if (straightWiresShell)
    XtPopdown(straightWiresShell);
  if (nodeCoordShell)
    XtPopdown(nodeCoordShell);
  XtPopdown(shell);

  w = NULL;
  cbs = NULL;
}

/*****************************************************************/
void editWireButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  Widget box, list;
  extern void openStraightWiresWindow();

  openStraightWiresWindow();
  box = XtNameToWidget (straightWiresShell,
     "straightWiresForm.straightWiresBox");
  list = XmSelectionBoxGetChild (box, XmDIALOG_LIST);
  XmListSetPos(list, wireIndex);
  XmListSelectPos(list, wireIndex, True);

  w = NULL;
  shell = NULL;
  cbs = NULL;
}

/*****************************************************************/
void editNodeButtonCB (w, node, cbs)
Widget w;
int node;
XmPushButtonCallbackStruct *cbs;
```

```
{
  int index;
  Widget list;
  extern Widget nodeCoordList;
  extern void openNodeCoordWindow();

  openNodeCoordWindow();
  list = nodeCoordList;
  if (node == 1)
    index = GW_END1[wireIndex - 1];
  else
    index = GW_END2[wireIndex - 1];
  XmListSetPos(list, index);
  XmListSelectPos(list, index, True);

  w = NULL;
  cbs = NULL;
}

/*****************************************************************/
void updateCB (w, drawInfo, event)
Widget w;
DrawData *drawInfo;
XEvent *event;
{
  extern void geometryFilter();

  if (event->xany.type == FocusOut || drawInfo->type >= 17)
    return;

  if (DRAW) {
    geometryFilter();
    initialize(drawInfo->source, drawInfo->freq, drawInfo->type);
    DrawGeo (drawInfo->type);
    DRAW = False;
  }
  w = NULL;
}

/*****************************************************************
 * When user closes the window, free memory.
 *****************************************************************/
static void destroyCB (widget)
    Widget widget;
{
  DrawData *drawData;

  XtVaGetValues (widget, XmNuserData, &drawData, NULL);
  /* The following is commented out because it keeps crashing!
  XgCloseDevice(drawData->xgd);
  */
  XtFree ((char *) drawData);
  XtDestroyWidget (widget);
  if (straightWiresShell)
    XtPopdown(straightWiresShell);
  if (nodeCoordShell)
    XtPopdown(nodeCoordShell);
  if (popupShell) {
    XtPopdown(popupShell);
    popupShell = NULL;
  }
} /* end destroyCB */
```

A.19 cEditGeo.c, fEditGeo.c:

cEditGeo.c:

```
/*
 * Callbacks for the Edit Geometry Cards window
 */

#include <Xm/List.h>
#include <Xm/Text.h>
#include "control.h"
#include "cFileMenu.h"

extern int getListPosition ();

extern struct glink *gcheckList, *gbufferList, *gposition[];
extern struct glink *gholdList;
extern struct gcountRecord grecord;
extern int siblingsOfWidget();
extern void gCopyCount();
extern void gResetCount();
extern void updateGeoData();
extern struct glink *gCopyList();
extern struct glink *gResetList();
extern struct glink *gEmptyList();
extern XmStringTable clearTable();

/*****************************************************************************/
void editGeoAddButtonCB (w, shell)
    Widget w, shell;
{
 char *string;
 XmString xmstring;
 int index;
 CardData *cardData;

 XtVaGetValues (shell, XmNuserData, &cardData, NULL);

 string = XmTextGetString (cardData->text);

 if (string[0] == '\0') {
  XtFree (string);
  return;
 }

 xmstring = XmStringCreateSimple (string);
 XtFree (string);

 /* Get the current list selection */
 index = getListPosition (cardData->list) + 1;

 /* Insert string into list */
 XmListAddItem (cardData->list, xmstring, index);
 XmListSelectPos (cardData->list, index, TRUE);
 XmStringFree (xmstring);

 w = w;  /* Make compiler happy */

} /* end editGeoAddButtonCB */

/*****************************************************************************/
void editGeoModifyButtonCB (w, shell)
    Widget w, shell;
{
 char *string;
 int index;
 XmString xmstring;
 CardData *cardData;

 XtVaGetValues (shell, XmNuserData, &cardData, NULL);

 string = XmTextGetString (cardData->text);

 if (string[0] == '\0') {
  XtFree (string);
  return;
 }

 xmstring = XmStringCreateSimple (string);
 XtFree (string);

 /* Get the current list selection */
 if (index = getListPosition (cardData->list)) {

  /* Replace string in list */
  XmListReplaceItemsPos (cardData->list, &xmstring, 1, index);
  XmListSelectPos (cardData->list, index, TRUE);
  XmStringFree (xmstring);
 }
 w = w;  /* Make compiler happy */
```

```
} /* end editGeoModifyButton */

/*****************************************************************************/
void editGeoDeleteButtonCB (w, shell)
    Widget w, shell;
{
 int index;
 CardData *cardData;
 int count, *positionList;

 XtVaGetValues (shell, XmNuserData, &cardData, NULL);

 /* Get the current list selection */
 if (index = getListPosition (cardData->list)) {

  /* Delete string in list */
  XmListGetSelectedPos (cardData->list, &positionList, &count);
  XmListDeletePositions (cardData->list, positionList, count);
  XtFree ((char *)positionList);
  XmListSelectPos (cardData->list, index, TRUE);
  if (!XmListPosSelected(cardData->list, index))
   XmListSelectPos (cardData->list, index - 1, TRUE);
 }
 w = w;  /* Make compiler happy */

} /* end editGeoDeleteButton */

/*****************************************************************************/
void editGeoCopyButtonCB (w, shell)
    Widget w, shell;
{
 int index;
 XmString *items, test[3];
 CardData *cardData;
 int i, j, count, *positionList;
 int *pos, poscount = 0;

 XtVaGetValues (shell, XmNuserData, &cardData, NULL);

 /* Get the current list selection */
 if (index = getListPosition (cardData->list)) {

  /* Get the items */
  XtVaGetValues (cardData->list, XmNitems, &items, NULL);

  XtVaGetValues (cardData->list, XmNselectedItems, &items, NULL);
  XmListGetSelectedPos (cardData->list, &positionList, &count);

  pos = (int *)XtMalloc(count * sizeof(int));
  test[0] = XmStringCreateSimple ("GM");
  test[1] = XmStringCreateSimple ("GR");
  test[2] = XmStringCreateSimple ("GX");
  for (i = 0; i < count; i++) {
   for (j = 0; j < 3; j++)
    if (XmStringHasSubstring(items[i], test[j]))
     break;
   if (j == 3) { /* not found */
    pos[poscount] = positionList[i];
    poscount++;
   }
  }
  for (i = 0; i < 3; i++)
   XmStringFree(test[i]);
  if (poscount) {
   for (i = 0; i < poscount; i++)
    XmListDeselectPos(cardData->list, pos[i]);
   XtFree ((char *)pos);
   XtFree ((char *)positionList);
   count = 0;
   XtVaGetValues (cardData->list, XmNselectedItems, &items, NULL)
;
   XmListGetSelectedPos (cardData->list, &positionList, &count);
   if (!count) {
    return;
   }
  }
  XtFree ((char *)positionList);
 }
 w = w;  /* Make compiler happy */

} /* end editGeoCopyButtonCB */

/*****************************************************************************/
void editGeoCutButtonCB (w, shell)
    Widget w, shell;
{
 int index;
 XmString *items, test[3];
 CardData *cardData;
 int i, j, count, *positionList;
 int *pos, poscount = 0;
 extern struct glink *gCutUpdate();
```

```
    XtVaGetValues (shell, XmNuserData, &cardData, NULL);

    /* Get the current list selection */
    if (index = getListPosition (cardData->list)) {

     XtVaGetValues (cardData->list, XmNselectedItems, &items, NULL);
     XmListGetSelectedPos (cardData->list, &positionList, &count);

     pos = (int *)XtMalloc(count * sizeof(int));
     test[0] = XmStringCreateSimple ("GM");
     test[1] = XmStringCreateSimple ("GR");
     test[2] = XmStringCreateSimple ("GX");
     for (i = 0; i < count; i++) {
      for (j = 0; j < 3; j++)
       if (XmStringHasSubstring(items[i], test[j]))
        break;
      if (j == 3) { /* not found */
       pos[poscount] = positionList[i];
       poscount++;
      }
     }
     for (i = 0; i < 3; i++)
      XmStringFree(test[i]);
     if (poscount) {
      createMessageDialog(w, "Message",
      "Only GM, GR, GX will be cut", 0);
      for (i = 0; i < poscount; i++)
       XmListDeselectPos(cardData->list, pos[i]);
      XtFree ((char *)pos);
      XtFree ((char *)positionList);
      count = 0;
      XtVaGetValues (cardData->list, XmNselectedItems, &items, NULL);
      XmListGetSelectedPos (cardData->list, &positionList, &count);
      if (!count) {
       return;
      }
     }
     gbufferList = gCutUpdate(cardData->list, gposition,
                  gcheckList, gbufferList);
     if (!gbufferList)
      return;

     XmListDeletePositions (cardData->list, positionList, count);
     XtFree ((char *)positionList);

     /* Delete string in list */
     XmListSelectPos (cardData->list, index, TRUE);
     if (!XmListPosSelected(cardData->list, index))
      XmListSelectPos (cardData->list, index - 1, TRUE);
    }
    w = w; /* Make compiler happy */

} /* end editGeoCutButtonCB */

/*****************************************************************************/
void editGeoPasteButtonCB (w, shell)
    Widget w, shell;
{
  int index;
  CardData *cardData;
  int i = 0;
  struct glink *node;
  extern void gPasteUpdate();

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  /* Get the current list selection */
  gPasteUpdate(cardData->list, gposition, gcheckList, gbufferList);

   /* Insert string in cutBuffer into current position */
  index = getListPosition (cardData->list);
  if (index == 0) index = 1;
  node = gbufferList;
  while (node) {
   XmListAddItem (cardData->list, node->string, index + i);
   node = node->next;
   i++;
  }
  XmListDeselectAllItems (cardData->list);
  XmListSelectPos (cardData->list, index, TRUE);

  w = w; /* Make compiler happy */

} /* end editGeoPasteButtonCB */

/*****************************************************************************/
void editGeoOkButtonCB (w, pane)
    Widget w, pane;
{
  extern void editGeoApplyButtonCB ();
  CardData *cardData;

  editGeoApplyButtonCB (w, pane);
```

```
  XtVaGetValues (pane, XmNuserData, &cardData, NULL);
  XtPopdown (*(cardData->shell));
} /* end editGeoOkButtonCB */

/*****************************************************************************/
void editGeoApplyButtonCB (w, shell)
    Widget w, shell;
{
  Arg args [2];
  XmString *items;
  int i, itemCount;
  extern Widget momExportList;
  extern XmString *GeometryCards, *ControlCards;
  extern int NumGeometryCards, NumControlCards;
  CardData *cardData;
  extern Boolean saveAlert;

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  /* Get list items & item count */
  XtSetArg (args[0], XmNitems, &items);
  XtSetArg (args[1], XmNitemCount, &itemCount);
  XtGetValues (cardData->list, args, 2);

  /* Deallocate then allocate room for cards */
  for (i = 0; i < cardData->numCards; i++)
    XmStringFree (cardData->cards[i]);
  XtFree ((char *) cardData->cards);
  cardData->cards = (XmString *) XtMalloc (sizeof (XmString) * itemCount);

  /* Copy the items and itemCount */
  for (i = 0; i < itemCount; i++)
    cardData->cards[i] = XmStringCopy (items [i]);
  cardData->numCards = itemCount;

  /* Update global variables */
  if (cardData->type == 0) {
    GeometryCards = cardData->cards;
    NumGeometryCards = itemCount;
  } else {
    ControlCards = cardData->cards;
    NumControlCards = itemCount;
  }

  gholdList = gCopyList (gcheckList, gholdList);
  gCopyCount(&grecord);
  saveAlert = True;

  w = w;  /* Make compiler happy */

} /* end editGeoApplyButtonCB */

/*****************************************************************************/
void editGeoResetButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  Arg args [2];
  CardData *cardData;

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  XtSetArg (args[0], XmNitems, cardData->cards);
  XtSetArg (args[1], XmNitemCount, cardData->numCards);
  XtSetValues (cardData->list, args, 2);
  XmListDeselectAllItems(cardData->list);

  gcheckList = gResetList (gcheckList, gholdList, gposition);
  gResetCount(&grecord);
  updateGeoData (gcheckList, aLL);
  gbufferList = gEmptyList(gbufferList);
  XmListDeselectAllItems(cardData->list);

  w = w;  /* Make compiler happy */
  cbs = NULL;

} /* end editGeoResetButtonCB */

/*****************************************************************************/
void editGeoCancelButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  extern Widget getTopShell();

  editGeoResetButtonCB(w, shell, cbs);
  shell = getTopShell(shell);
  XtPopdown (shell);
}
```

fEditGeo.c:

```
/*
 * Procedures for creating the Edit Geometry Cards Window
 */

#include "control.h"
#include <Xm/Form.h>
#include <Xm/List.h>
#include <Xm/PanedW.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/TextF.h>
#include <stdio.h>
#include "actionArea.h"
#include "ctrlgeo.h"

extern Widget topLevel;
Widget editGeoShell = NULL;
extern Widget editCtrlShell;

extern void editGeoOkButtonCB ();
extern void editGeoApplyButtonCB ();
extern void editGeoResetButtonCB ();
static void destroyCB ();
Widget createEditWindow ();

extern Boolean isPoppedUp ();
extern void editGeoCancelButtonCB ();

Widget geoList = NULL;

/*****************************************************************/
void openEditWindow ()
{
 Widget shell;
 Boolean notOpen = FALSE;
 CardData *cardData;
 extern XmString  *GeometryCards;
 extern int NumGeometryCards;
 extern struct glink *gcheckList, *gbufferList;
 extern void setGeoList();
 extern XmString *updateCards();
 extern struct glink *gEmptyList();

 if (editGeoShell == NULL) {
  editGeoShell = createEditWindow ("Edit Card Order", GeometryCards,
                   NumGeometryCards, 0);
  notOpen = TRUE;
 }

 shell = editGeoShell;
 if (!isPoppedUp(shell))
  notOpen = TRUE;

 if (notOpen) {
  Widget pane = XtNameToWidget (shell, "pane");
  XtVaGetValues (pane, XmNuserData, &cardData, NULL);
  setGeoList(cardData->list, gcheckList);
  GeometryCards = updateCards(cardData->list,
                   GeometryCards, &NumGeometryCards);
  cardData->cards = GeometryCards;
  cardData->numCards = NumGeometryCards;
  gbufferList = gEmptyList(gbufferList);
  XtPopup (shell, XtGrabNone);
 }
} /* end openEditWindow */

/*****************************************************************/
Widget createEditWindow (name, cards, numCards, cardtype)
 char *name;
 XmString *cards;
 int numCards, cardtype;
{
 Widget shell, form, pane, list, rowColumn, button;
 Arg args [20];
 int n = 0;
 CardData *cardData;
 extern void editGeoAddButtonCB ();
 extern void editGeoModifyButtonCB ();
 extern void editGeoDeleteButtonCB ();
 extern void editGeoCutButtonCB ();
 extern void editGeoPasteButtonCB ();
 static ActionAreaItem actionItems [] = {
  {"Ok", editGeoOkButtonCB, NULL},
  {"Apply", editGeoApplyButtonCB, NULL},
  {"Reset", editGeoResetButtonCB, NULL},
  {"Cancel", editGeoCancelButtonCB, NULL}
 };
 extern void newSelectActionTable ();
 extern void editGeoCopyButtonCB ();
```

```
extern void newEscapeAction();
extern Widget momExportShell;

XtSetArg (args[0], XmNdeleteResponse, XmDESTROY);
shell = XtCreatePopupShell (name, topLevelShellWidgetClass,
                topLevel, args, 1);

newEscapeAction(shell);

/* Create structure for storing card data */
cardData = (CardData *) XtMalloc (sizeof (CardData));
cardData->shell = cardtype == 0 ? &editGeoShell : &editCtrlShell;
cardData->cards = cards;
cardData->numCards = numCards;
cardData->type = cardtype;

XtSetArg (args[0], XmNsashWidth, 1);
XtSetArg (args[1], XmNsashHeight, 1);
pane = XmCreatePanedWindow (shell, "pane", args, 2);
XtVaSetValues (pane, XmNuserData, cardData, NULL);
XtAddCallback (shell, XmNdestroyCallback, destroyCB, pane);
form = XmCreateForm (pane, "form", args, 0);

n = 0;
XtSetArg (args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNrightOffset, 15); n++;
XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNtopOffset, 15); n++;
XtSetArg (args[n], XmNisAligned, True); n++;
XtSetArg (args[n], XmNentryAlignment, XmALIGNMENT_CENTER); n++;
rowColumn = XmCreateRowColumn (form, "rowColumn", args, n);
button = XmCreatePushButton (rowColumn, "Cut", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNactivateCallback, editGeoCutButtonCB, pane);
button = XmCreatePushButton (rowColumn, "Paste", args, 0);
XtManageChild (button);
XtAddCallback (button, XmNactivateCallback, editGeoPasteButtonCB, pane);

XtManageChild (rowColumn);

/* Create list box for holding cards */
n = 0;
XtSetArg (args[n], XmNrightAttachment, XmATTACH_WIDGET); n++;
XtSetArg (args[n], XmNrightWidget, rowColumn); n++;
XtSetArg (args[n], XmNrightOffset, 10); n++;
XtSetArg (args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNleftOffset, 15); n++;
XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNtopOffset, 15); n++;
XtSetArg (args[n], XmNvisibleItemCount, 20); n++;
XtSetArg (args[n], XmNwidth, 700); n++;
list = XmCreateScrolledList (form, "list", args, n);
cardData->list = list;
XtManageChild (list);

geoList = list;

n=0;
XtSetArg(args[n], XmNselectionPolicy, XmEXTENDED_SELECT); n++;
XtSetValues(list, args, n);
newSelectActionTable (list);

XtManageChild (form);

/* Create action area at bottom of window */
actionItems[0].data = (XtPointer) pane;
actionItems[1].data = (XtPointer) pane;
actionItems[2].data = (XtPointer) pane;
actionItems[3].data = (XtPointer) pane;

createActionArea (pane, actionItems, XtNumber (actionItems));
XtManageChild (pane);

return (shell);

} /* end createSelectionBox */

static void destroyCB (w, pane)
    Widget w, pane;
{
 CardData  *cardData;

 XtVaGetValues (pane, XmNuserData, &cardData, NULL);
 *(cardData->shell) = NULL;
 XtFree ((char *) cardData);

 w = w; /* Make compiler happy */

} /* end destroyCB */
```

A.20 cNeedControl.c, fNeedControl.c:

cNeedControl.c:

```
/*
 * Callbacks for the Edit Geometry Cards window
 */

#include "control.h"
#include <Xm/List.h>
#include <Xm/Text.h>
#include "cFileMenu.h"

extern int getListPosition ();

Boolean replaceNotify = False;

extern struct link *checkList, *bufferList, *position[];
extern struct link *holdList;
extern void copyCount();
extern void resetCount();
extern void updateControlData();
extern struct link *copyList();
extern struct link *resetList();
extern struct link *emptyList();
extern XmStringTable clearTable();
extern struct countRecord record;

/*****************************************************************************/
void optionToWindow(optionlist)
Widget optionlist;
{
 extern void openFrequencyWindow ();
 extern void openLoadsWindow ();
 extern void openVoltageSourcesWindow ();
 extern void openIncidentPlaneWaveWindow ();
 extern void openTransmissionLinesWindow ();
 extern void openTwoPortNetsWindow ();
 extern void openInsulatedWiresWindow ();
 extern void openGroundParamWindow ();
 extern void openAddGroundParamWindow ();
 extern void openUpperMediumParamWindow ();
 extern void openMaxCouplingWindow ();
 extern void openNearElectricWindow ();
 extern void openNearMagneticWindow ();
 extern void openRadiationPatternWindow ();
 extern void openPrintOptionsWindow ();
 int index;

 index = getListPosition (optionlist);
 switch (index) {
  case 1:  openFrequencyWindow ();
           break;
  case 2:  openLoadsWindow ();
           break;
  case 3:  openVoltageSourcesWindow ();
           break;
  case 4:  openIncidentPlaneWaveWindow ();
           break;
  case 5:  openTransmissionLinesWindow ();
           break;
  case 6:  openTwoPortNetsWindow ();
           break;
  case 7:  openInsulatedWiresWindow ();
           break;
  case 8:  openGroundParamWindow ();
           break;
  case 9:  openAddGroundParamWindow ();
           break;
  case 10: openUpperMediumParamWindow ();
           break;
  case 11: openMaxCouplingWindow ();
           break;
  case 12: openNearElectricWindow ();
           break;
  case 13: openNearMagneticWindow ();
           break;
  case 14: openRadiationPatternWindow ();
           break;
  case 15: openPrintOptionsWindow (PQ);
           break;
  case 16: openPrintOptionsWindow (PS);
           break;
  case 17: openPrintOptionsWindow (PT);
           break;
  default:
           break;
 }
} /* end optionToWindow */

/*****************************************************************************/
```

```
void descriptionToWindow(descriptionlist)
Widget descriptionlist;
{
 extern void openFrequencyWindow ();
 extern void openLoadsWindow ();
 extern void openVoltageSourcesWindow ();
 extern void openIncidentPlaneWaveWindow ();
 extern void openTransmissionLinesWindow ();
 extern void openTwoPortNetsWindow ();
 extern void openInsulatedWiresWindow ();
 extern void openGroundParamWindow ();
 extern void openAddGroundParamWindow ();
 extern void openUpperMediumParamWindow ();
 extern void openMaxCouplingWindow ();
 extern void openNearElectricWindow ();
 extern void openNearMagneticWindow ();
 extern void openRadiationPatternWindow ();
 extern void openPrintOptionsWindow ();
 int index;
 struct link *node;

 index = getListPosition(descriptionlist);
 if (!index)
  return;

 node = position[index];
 /* save the order at the head */
 position[0]->order = node->order;
 switch (node->tableType) {
  case FR:  openFrequencyWindow ();
        break;
  case LD:  openLoadsWindow ();
        break;
  case EX0: openVoltageSourcesWindow ();
        break;
  case EX1: openIncidentPlaneWaveWindow ();
        break;
  case TL:  openTransmissionLinesWindow ();
        break;
  case NT:  openTwoPortNetsWindow ();
        break;
  case IS:  openInsulatedWiresWindow ();
        break;
  case GN:  openGroundParamWindow ();
        break;
  case GD:  openAddGroundParamWindow ();
        break;
  case UM:  openUpperMediumParamWindow ();
        break;
  case CP:  openMaxCouplingWindow ();
        break;
  case NE:  openNearElectricWindow ();
        break;
  case NH:  openNearMagneticWindow ();
        break;
  case RP:  openRadiationPatternWindow ();
        break;
  case PQ:  openPrintOptionsWindow (PQ);
        break;
  case PS:  openPrintOptionsWindow (PS);
        break;
  case PT:  openPrintOptionsWindow (PT);
        break;
  default:
        break;
 }
} /* end descriptionToWindow */

/*****************************************************************************/
void editCtrlAddButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
 int index;
 CardData *cardData;
 char newstring[132];
 XmString xmString;
 extern Widget controlList;
 struct link *node, *next, *prev;
 extern void updatePosition();

 XtVaGetValues (shell, XmNuserData, &cardData, NULL);

 replaceNotify = False;
 optionToWindow(cardData->optionlist);

 if (getListPosition(cardData->optionlist) == 18) {
  sprintf (newstring, "XQ");
  xmString = XmStringCreateSimple (newstring);
  node = (struct link *)XtMalloc(sizeof(struct link));
  node->tableType = XQ;
  node->string = XmStringCopy(xmString);
  index = getListPosition (controlList);
```

```
    if (!index)  index = 1;
    XmListAddItemUnselected (controlList, xmString, index);
    XmStringFree (xmString);

    prev = position[index - 1];
    next = position[index];
    prev->next = node;
    node->prev = prev;
    node->next = next;
    next->prev = node;
    updatePosition(position, checkList);
    ExecuteCount++;
  }

  w = w;  /* Make compiler happy */
  cbs = NULL;

} /* end editCtrlAddButtonCB */

/*****************************************************************************/
void editCtrlModifyButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  CardData *cardData;

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  replaceNotify = True;
  descriptionToWindow(cardData->descriptionlist);

  w = w;  /* Make compiler happy */
  cbs = NULL;

} /* end editCtrlModifyButton */

/*****************************************************************************/
void editCtrlDeleteButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  int index;
  CardData *cardData;
  int count = 0, *positionList = NULL;
  XmString item;
  extern void deleteUpdate();

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  /* Get the current list selection */
  if (index = getListPosition (cardData->descriptionlist)) {

    item = XmStringCreateSimple ("EN");
    XmListDeselectItem(cardData->descriptionlist, item);
    XmStringFree(item);

    deleteUpdate(cardData->descriptionlist, position, checkList);

    if (XmListGetSelectedPos (cardData->descriptionlist, &positionList, &count)) {
      XmListDeletePositions (cardData->descriptionlist, positionList, count);
      XtFree ((char *)positionList);
    }
    XmListSelectPos (cardData->descriptionlist, index, TRUE);
    if (!XmListPosSelected(cardData->descriptionlist, index))
      XmListSelectPos (cardData->descriptionlist, index - 1, TRUE);
  }
  w = w;  /* Make compiler happy */
  cbs = NULL;

} /* end editCtrlDeleteButtonCB */

/*****************************************************************************/
void editCtrlCopyButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  int index;
  CardData *cardData;
  XmString item;
  extern struct link *copyUpdate();

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  /* Get the current list selection */
  if (index = getListPosition (cardData->descriptionlist)) {

  item = XmStringCreateSimple ("EN");
  XmListDeselectItem(cardData->descriptionlist, item);
  XmStringFree(item);

  bufferList = copyUpdate(cardData->descriptionlist, position,
                bufferList);
  if (!bufferList)
```

```
    return;
  }
  w = w;  /* Make compiler happy */
  cbs = NULL;

} /* end editCtrlCopyButtonCB */

/*****************************************************************/
void editCtrlCutButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  int index;
  CardData *cardData;
  int count = 0, *positionList = NULL;
  XmString item;
  extern struct link *cutUpdate();

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  /* Get the current list selection */
  if (index = getListPosition (cardData->descriptionlist)) {

  item = XmStringCreateSimple ("EN");
  XmListDeselectItem(cardData->descriptionlist, item);
  XmStringFree(item);

  bufferList = cutUpdate(cardData->descriptionlist, position,
            checkList, bufferList);
  if (!bufferList)
    return;

    if (XmListGetSelectedPos (cardData->descriptionlist, &positionList, &count)) {
      XmListDeletePositions (cardData->descriptionlist, positionList, count);
      XtFree ((char *)positionList);
    }
  }
  w = w;  /* Make compiler happy */
  cbs = NULL;

} /* end editCtrlCutButtonCB */

/*****************************************************************/
void editCtrlPasteButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
  int index;
  CardData *cardData;
  int i = 0, count = 0, *positionList = NULL;
  struct link *node;
  extern void pasteUpdate();

  XtVaGetValues (shell, XmNuserData, &cardData, NULL);

  /* Get the current list selection */
  if (XmListGetSelectedPos (cardData->descriptionlist, &positionList, &count)) {

    pasteUpdate(cardData->descriptionlist, position, checkList,
          bufferList);

    /* Insert string in cutBuffer into current position */
    index = positionList[0];
    XtFree ((char *)positionList);
    node = bufferList;
    while (node) {
      XmListAddItem (cardData->descriptionlist, node->string, index + i);
      node = node->next;
      i++;
    }
    XmListDeselectAllItems (cardData->descriptionlist);
    XmListSelectPos (cardData->descriptionlist, index + i, FALSE);
  }
  w = w;  /* Make compiler happy */
  cbs = NULL;

} /* end editCtrlPasteButtonCB */

/*****************************************************************/
void editCtrlOkButtonCB (w, pane, cbs)
    Widget w, pane;
    XmPushButtonCallbackStruct *cbs;
{
  extern void editCtrlApplyButtonCB ();
  CardData *cardData;

  editCtrlApplyButtonCB (w, pane, cbs);

  XtVaGetValues (pane, XmNuserData, &cardData, NULL);
  XtPopdown (*(cardData->shell));
}
/*****************************************************************/
void editCtrlApplyButtonCB (w, shell, cbs)
```

```
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
 Arg args [2];
 XmString *items;
 int i, itemCount = 0;
 extern Widget momExportList;
 extern XmString *GeometryCards, *ControlCards;
 extern int NumGeometryCards, NumControlCards;
 CardData *cardData;
 extern Boolean saveAlert;

 XtVaGetValues (shell, XmNuserData, &cardData, NULL);

 /* Get list items & item count */
 XtSetArg (args[0], XmNitems, &items);
 XtSetArg (args[1], XmNitemCount, &itemCount);
 XtGetValues (cardData->descriptionlist, args, 2);

 /* Deallocate then allocate room for cards */
 for (i = 0; i < cardData->numCards; i++)
  XmStringFree (cardData->cards[i]);
 XtFree ((char *) cardData->cards);
 cardData->cards = (XmString *) XtMalloc (sizeof (XmString) * itemCount);

 /* Copy the items and itemCount */
 for (i = 0; i < itemCount; i++)
  cardData->cards[i] = XmStringCopy (items [i]);
 cardData->numCards = itemCount;

 /* Update global variables */
 if (cardData->type == 0) {
  GeometryCards = cardData->cards;
  NumGeometryCards = itemCount;
 } else {
  ControlCards = cardData->cards;
  NumControlCards = itemCount;
 }

 holdList = copyList (checkList, holdList);
 copyCount(&record);
 saveAlert = True;

 w = NULL;
 cbs = NULL;

} /* end editCtrlApplyButtonCB */
/*****************************************************************************/
void editCtrlResetButtonCB (w, shell, cbs)
Widget w, shell;
XmPushButtonCallbackStruct *cbs;
{
 Arg args [2];
 CardData *cardData;

 XtVaGetValues (shell, XmNuserData, &cardData, NULL);

 XtSetArg (args[0], XmNitems, cardData->cards);
 XtSetArg (args[1], XmNitemCount, cardData->numCards);
 XtSetValues (cardData->descriptionlist, args, 2);

 checkList = resetList (checkList, holdList, position);
 resetCount(&record);
 updateControlData (checkList, ALL);
 bufferList = emptyList(bufferList);
 XmListDeselectAllItems(cardData->descriptionlist);

 w = w;  /* Make compiler happy */
 cbs = NULL;

} /* end editCtrlResetButtonCB */

/*****************************************************************************/
void editCtrlCancelButtonCB (w, shell, cbs)
   Widget w, shell;
   XmPushButtonCallbackStruct *cbs;
{
 extern Widget getTopShell();

 editCtrlResetButtonCB(w, shell, cbs);
 shell = getTopShell(shell);
 XtPopdown (shell);
} /* end editCtrlCancelButtonCB */
```

fNeedControl.c:

```
/*
 * Procedures for creating the Edit Control Cards Window
 */
```

```
#include "control.h"
#include <Xm/Form.h>
#include <Xm/List.h>
#include <Xm/PanedW.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/TextF.h>
#include <Xm/Label.h>
#include <stdio.h>
#include "actionArea.h"
#include "ctrlgeo.h"

extern Widget topLevel;
extern Widget editGeoShell;
Widget editCtrlShell = NULL;
Widget controlList;

extern void editCtrlOkButtonCB ();
extern void editCtrlApplyButtonCB ();
extern void editCtrlResetButtonCB ();
extern void editCtrlCancelButtonCB ();
Widget createEditCtrlWindow ();
static void destroyCB ();

extern Boolean isPoppedUp ();

/*****************************************************************************/
XmString *updateCards(list, cards, numcards)
Widget list;
XmString *cards;
int *numcards;
{
  Arg args [2];
  XmString *items;
  int i, itemCount = 0;

  /* Get list items & item count */
  XtSetArg (args[0], XmNitems, &items);
  XtSetArg (args[1], XmNitemCount, &itemCount);
  XtGetValues (list, args, 2);
  /* Deallocate then allocate room for cards */
  for (i = 0; i < *numcards; i++)
    XmStringFree (cards[i]);
  XtFree ((char *)cards);

  if (!itemCount)
    return NULL;

  cards = (XmString *) XtMalloc (sizeof (XmString) * itemCount);
  /* Copy the items and itemCount */
  for (i = 0; i < itemCount; i++)
    cards[i] = XmStringCopy (items[i]);
  *numcards = itemCount;

  return cards;
} /* end updateCards */

/*****************************************************************************/
void openEditCtrlWindow ()
{
  Widget shell;
  Boolean notOpen = FALSE;
  CardData *cardData;
  extern XmString *ControlCards;
  extern int NumControlCards;
  extern struct link *checkList, *bufferList;
  extern void setControlList();
  extern struct link *emptyList();

  if (editCtrlShell == NULL) {
    editCtrlShell = createEditCtrlWindow ("Edit Control Cards", ControlCards,
                                          NumControlCards, 1);
    notOpen = TRUE;
  }

  shell = editCtrlShell;
  if (!isPoppedUp(shell))
    notOpen = TRUE;

  if (notOpen) {
    Widget pane = XtNameToWidget (shell, "pane");
    XtVaGetValues (pane, XmNuserData, &cardData, NULL);
    setControlList(cardData->descriptionlist, checkList);
    ControlCards = updateCards(cardData->descriptionlist,
                               ControlCards, &NumControlCards);
    cardData->cards = ControlCards;
    cardData->numCards = NumControlCards;
    bufferList = emptyList(bufferList);
    XtPopup (shell, XtGrabNone);
  }
} /* end openEditWindow */
```

```
/*****************************************************************/
Widget createEditCtrlWindow (name, cards, numCards, cardtype)
 char *name;
 XmString *cards;
 int numCards, cardtype;
{
 Widget shell, form, pane, label, list, rowColumn, button;
 Arg args [17];
 int n = 0;
 CardData *cardData;
 extern void editCtrlAddButtonCB ();
 extern void editCtrlModifyButtonCB ();
 extern void editCtrlDeleteButtonCB ();
 extern void editCtrlCutButtonCB ();
 extern void editCtrlPasteButtonCB ();
 static ActionAreaItem actionItems [] = {
  {"Ok", editCtrlOkButtonCB, NULL},
  {"Apply", editCtrlApplyButtonCB, NULL},
  {"Reset", editCtrlResetButtonCB, NULL},
  {"Cancel", editCtrlCancelButtonCB, NULL}
 };
 static char *options[] = {
                          "Frequency (FR)",
                          "Loads (LD)",
                          "Voltage Sources (EX)",
                          "Incident Plane Wave (EX)",
                          "Transmission Lines (TL)",
                          "Two Port Networks (NT)",
                          "Insulated Wire (IS)",
                          "Ground Parameters (GN)",
                          "Additional Ground Parameters (GD)",
                          "Upper Medium Parameters (UM)",
                          "Maximum Coupling (CP)",
                          "Near Electric Fields (NE)",
                          "Near Magnetic Fields (NH)",
                          "Radiation Patterns (RP)",
        "Print Options for Charge (PQ)",
        "Print Electrical Lengths (PS)",
        "Print Options for Current (PT)",
                          "Execute (XQ)"
                          };
 XmString optionItems[40];
 extern void newSelectActionTable ();
 extern void editCtrlCopyButtonCB ();
 extern void newEscapeAction();
 extern Widget momExportShell;

 XtSetArg (args[0], XmNdeleteResponse, XmDESTROY);
 shell = XtCreatePopupShell (name, topLevelShellWidgetClass,
                  topLevel, args, 1);

 newEscapeAction(shell);

 /* Create structure for storing card data */
 cardData = (CardData *) XtMalloc (sizeof (CardData));
 cardData->shell = cardtype == 0 ? &editGeoShell : &editCtrlShell;
 cardData->cards = cards;
 cardData->numCards = numCards;
 cardData->type = cardtype;

 XtSetArg (args[0], XmNsashWidth, 1);
 XtSetArg (args[1], XmNsashHeight, 1);
 pane = XmCreatePanedWindow (shell, "pane", args, 2);
 XtVaSetValues (pane, XmNuserData, cardData, NULL);
 XtAddCallback (shell, XmNdestroyCallback, destroyCB, pane);
 form = XmCreateForm (pane, "form", args, 0);

 n = 0;
 XtSetArg (args[n], XmNrightAttachment, XmATTACH_FORM); n++;
 XtSetArg (args[n], XmNrightOffset, 15); n++;
 XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
 XtSetArg (args[n], XmNtopOffset, 35); n++;
 XtSetArg (args[n], XmNisAligned, True); n++;
 XtSetArg (args[n], XmNentryAlignment, XmALIGNMENT_CENTER); n++;
 rowColumn = XmCreateRowColumn (form, "rowColumn", args, n);

 /* Create editing buttons */
 button = XmCreatePushButton (rowColumn, "Add", args, 0);
 XtManageChild (button);
 XtAddCallback (button, XmNactivateCallback, editCtrlAddButtonCB, pane);
 button = XmCreatePushButton (rowColumn, "Modify", args, 0);
 XtManageChild (button);
 XtAddCallback (button, XmNactivateCallback, editCtrlModifyButtonCB, pane);
 button = XmCreatePushButton (rowColumn, "Delete", args, 0);
 XtManageChild (button);
 XtAddCallback (button, XmNactivateCallback, editCtrlDeleteButtonCB, pane);

 button = XmCreatePushButton (rowColumn, "Copy", args, 0);
 XtManageChild (button);
 XtAddCallback (button, XmNactivateCallback, editCtrlCopyButtonCB, pane);

 button = XmCreatePushButton (rowColumn, "Cut", args, 0);
 XtManageChild (button);
```

```
    XtAddCallback (button, XmNactivateCallback, editCtrlCutButtonCB, pane);
    button = XmCreatePushButton (rowColumn, "Paste", args, 0);
    XtManageChild (button);
    XtAddCallback (button, XmNactivateCallback, editCtrlPasteButtonCB, pane);

    XtManageChild (rowColumn);

    /* Create list box for holding cards */
    n = 0;
    XtSetArg (args[n], XmNrightAttachment, XmATTACH_WIDGET); n++;
    XtSetArg (args[n], XmNrightWidget, rowColumn); n++;
    XtSetArg (args[n], XmNrightOffset, 10); n++;
    XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg (args[n], XmNtopOffset, 15); n++;
    XtSetArg (args[n], XmNentryAlignment, XmALIGNMENT_CENTER); n++;
    rowColumn = XmCreateRowColumn (form, "rowColumn1", args, n);
    label = XmCreateLabel(rowColumn,
          "                    Problem Description                    ",
        NULL, 0);
    XtManageChild(label);
    n = 0;
    XtSetArg(args[n], XmNselectionPolicy, XmEXTENDED_SELECT); n++;
    XtSetArg (args[n], XmNvisibleItemCount, 20); n++;
    list = XmCreateScrolledList (rowColumn, "list1", args, n);

    cardData->descriptionlist = list;
    controlList = list;
    XtManageChild (list);
    XtManageChild (rowColumn);

    n = 0;
    XtSetArg (args[n], XmNleftAttachment, XmATTACH_FORM); n++;
    XtSetArg (args[n], XmNleftOffset, 15); n++;
    XtSetArg (args[n], XmNrightAttachment, XmATTACH_WIDGET); n++;
    XtSetArg (args[n], XmNrightWidget, rowColumn); n++;
    XtSetArg (args[n], XmNrightOffset, 10); n++;
    XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
    XtSetArg (args[n], XmNtopOffset, 15); n++;
    XtSetArg (args[n], XmNisAligned, True); n++;
    XtSetArg (args[n], XmNentryAlignment, XmALIGNMENT_CENTER); n++;
    rowColumn = XmCreateRowColumn (form, "rowColumn2", args, n);
    label = XmCreateLabel(rowColumn, "Options", NULL, 0);
    XtManageChild(label);
    n = 0;
    XtSetArg (args[n], XmNvisibleItemCount, 20); n++;
    list = XmCreateScrolledList (rowColumn, "list2", args, n);
    cardData->optionlist = list;
    XtManageChild (list);
    XtManageChild (rowColumn);
    for (n = 0; n < XtNumber(options); n++)
      optionItems[n] = XmStringCreateSimple (options[n]);
    XtVaSetValues (list, XmNitems, optionItems,
            XmNitemCount, n, NULL);
    XmListSelectPos (list, 1, False);
    for (n = 0; n < XtNumber(options); n++)
      XmStringFree (optionItems[n]);

    XtManageChild (form);

    /* Create action area at bottom of window */
    actionItems[0].data = (XtPointer) pane;
    actionItems[1].data = (XtPointer) pane;
    actionItems[2].data = (XtPointer) pane;
    actionItems[3].data = (XtPointer) pane;
    createActionArea (pane, actionItems, XtNumber (actionItems));
    XtManageChild (pane);

    return (shell);

} /* end createSelectionBox */

static void destroyCB (w, pane)
    Widget w, pane;
{
    CardData *cardData;

    XtVaGetValues (pane, XmNuserData, &cardData, NULL);
    *(cardData->shell) = NULL;
    XtFree ((char *) cardData);

    w = w; /* Make compiler happy */

} /* end destroyCB */
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1995	Final: September 1995

4. TITLE AND SUBTITLE

NUMERICAL ELECTROMAGNETIC ENGINEERING DESIGN SYSTEM (NEEDS 3.1) WORKSTATION PROGRAMMER'S MANUAL

5. FUNDING NUMBERS

PE: 0603563N

6. AUTHOR(S)

J. C. Lam, J. W. Rockway, L. C. Russell, and D. T. Wentworth

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TD 2871

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Sea Systems Command (NAVSEA)
Arlington, VA 22242–5160

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This manual is a programmer's guide to the Numerical Electromagnetic Engineering Design System (NEEDS) Version 3.1 software program. NEEDS was developed to assist users of the Numerical Electromagnetics Code – Method of Moment (NEC–MoM). NEC–MoM requires rigidly defined inputs and often generates massive quantities of output. NEEDS makes NEC–MoM less tedious and more error-free. NEEDS 3.1 is specifically designed for NEC–MoM Version 4 users.

14. SUBJECT TERMS

Numerical Electromagnetic Engineering Design System (NEEDS)
Numerical Electromagnetics Code – Method of Moments (NEC–MoM)
Simple Raster Graphics Package (SRGP)
NEEDS programming source code

15. NUMBER OF PAGES

209

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
L. C. Russell	(619) 553-6132	Code 851

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0271	Archive/Stock	(6)
Code 0274	Library	(2)
Code 80	K. D. Regan	(1)
Code 8505	J. W. Rockway	(30)
Code 851	L. C. Russell	(1)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363–5100

Center for Naval Analyses
Alexandria, VA 22302–0268

Navy Acquisition, Research and Development
Information Center (NARDIC)
Arlington, VA 22244–5114

GIDEP Operations Center
Corona, CA 91718–8000

Naval Sea Systems Command
Arlington, VA 22242–5160 (3)